A Portuguese Flora Identification Tool Using Deep Learning

Miguel Ângelo Ribeiro Marques
Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos
Departamento de Ciências dos Computadores
2020

Orientador
Luís Lopes, Professor Associado, Faculdade de Ciências da Universidade do Porto

Coorientador
Eduardo Marques, Professor Auxiliar, Faculdade de Ciências da Universidade do Porto

Abstract

In today's world, deep learning (DL) models for image classification are standard in many areas, including for the identification of flora. Several models are used for this purpose, but most focus on flora spread all around the world. In this work, we present DL models specialised for the identification of autochthonous Portuguese flora. To build the models, we gathered images from Flora-On, a web portal for Portuguese flora by Sociedade Portuguesa de Botânica, and iNaturalist, the well-known site for crowd-sourced citizen science identifications of fauna and flora by the California Academy of Sciences and the National Geographic Society. We assembled three datasets, containing between 16,000 and 36,000 images and covering between 600 and 1,100 species. We then used the Google Auto ML cloud service to derive TensorFlow Lite models for each dataset. The models were then deployed in a web application that allows their comparative assessment. We compared the performance of the three models using three test groups, and the model with better performance was further compared to the state-of-the-art model used by PlantNet, a well-known site for crowd-sourced citizen science specialised in flora, showing promising results.

Keywords: biodiversity, flora, deep learning, image classification, citizen science

Resumo

No mundo de hoje, os modelos de deep learning (DL) para classificação de imagens são aplicados em muitas áreas, incluindo a identificação da flora. Vários modelos são usados para esse fim, mas a maioria abrange a flora de todo o mundo. Neste trabalho, apresentamos modelos de DL especializados na identificação da flora autóctone portuguesa. Para construir os modelos, reunimos imagens do Flora-On, um portal Web para a flora portuguesa da Sociedade Portuguesa de Botânica, e do iNaturalist, um conhecido projeto de ciência-cidadã global que regista avistamentos de fauna e de flora criado pela California Academy of Sciences e pela National Geographic Society. Construímos três conjuntos de dados, contendo entre 16.000 e 36.000 imagens e cobrindo entre 600 e 1.100 espécies. Em seguida, usamos o serviço Google Auto ML para derivar modelos do TensorFlow Lite para cada conjunto de dados. Os modelos foram então integrados numa aplicação Web para uso público generalizado. Comparamos depois o desempenho dos três modelos usando três grupos de teste, e o modelo com melhor desempenho foi ainda comparado ao modelo de última geração usado pelo PlantNet, um conhecido site de ciência-cidadã especializada em flora, obtendo resultados promissores.

Acknowledgements

I would like to express my thanks to Henrique Nepomuceno Alves (Parque Biológico de Gaia), for guidance and for providing the dataset for testing the models from his own photographic records, and to Miguel Porto (Sociedade Portuguesa de Botânica/Flora-On), for generously providing us with the Flora-On dataset. This work was partially funded by project SafeCities (POCI-01-0247-FEDER-041435) from Fundo Europeu de Desenvolvimento Regional (FEDER), through COMPETE 2020 and Portugal 2020. On a more personal note, I would like to thank my mom and my brothers for always supporting me. I also want to thank my friends who have always been there for me. Last but not least, a special thanks to my teachers, Luís Miguel Barros Lopes and Eduardo Resende Brandão Marques, for all the help provided and availability.

Dedicated to my friends and family.

Contents

List of Tables

List of Figures

Listings

Acronyms

AMLV	AutoML Vision
ANN	Artificial Neural Network
BLOB	Binary Large Object
BNN	Biological Neural Network
CNN	Convolutional Neural Network
CSV	Comma-separated values
CV	Computer Vision
DCC	Computer Science Department
DL	Deep Learning
DNN	Deep Neural Network
FN	False Negative
FP	False Positive
FCL	Fully Connected Layer
FCUP	Faculty of Sciences of the University of Porto
GBIF	Global Biodiversity Information Facility
GBUP	Galeria da Biodiversidade da Universidade do Porto
GCS	Google Cloud Storage
ITIS	Integrated Taxonomic Information System
ML	Machine Learning
NN	Neural Network
PBG	Parque Biológico de Gaia
ReLU	Rectified Linear Unit
TSN	Taxonomic Serial Number
TP	True Positive

Chapter 1

Introduction

Even though there are countless species on our planet, many still unknown to science [44], how many plants or animals can the average person recognize, let alone name? Some live right next to us while others can only be seen in other parts of the world. Whatever the case, all of them are part of the biodiversity that fills this world with unique places and landscapes. Biodiversity can be perceived as the variation of life at all levels of biological organization [7], from the simplest forms of life like bacteria, all the way to complete ecosystems. This includes plants, animals, fungi, and microbial life forms, and all of them contribute to the welfare of their respective ecosystems, which are essential to life on earth.

Throughout history, biodiversity has taken many hits, and now more than ever [28], changing weather patterns, pollution, invasive species, and over-exploitation pose as the main threats to its balance [18]. With all of this happening, it is mandatory the monitoring of species to measure the health of ecosystems, which is not an easy task to accomplish due to the lack of financial and human resources available to scientists. For this to work, it is necessary to sensitize and educate people to contribute with much-needed information and help scientists in this task. In recent years, a number of apps and web services to collect sightings and for automatic identification of flora species have appeared. Most services available can identify a wide range of species all across the globe. These services help people identify species through image recognition, using machine learning techniques, more specifically, deep learning.

1.1 Problem Statement

Work Proposal and Objectives

The Parque Biológico de Gaia (PBG) is an institution under the jurisdiction of the Câmara Municipal de Gaia, Established in 1983, with the focus on the preservation of local biodiversity and a total area of around 87 acres, the institution is host to many exhibitions and is an active participant in a variety of projects involving the different aspects of environmental protection

[27].

One project in specific, developed in association with the Department of Computer Science of the Faculty of Sciences of the University of Porto FCUP, was a citizen science mobile app, called Gaia, as well as a web server, to monitor the local biodiversity [37]. To engage the large numbers of users required for an effective monitorization such an app must provide interesting feedback. Automatic identification of the animal or plant being observed/photographed is highly regarded by users and provides an instant reward.

Our goal in this dissertation is to build upon this earlier work and to introduce in it key user reward mechanism, through a web service that automatically identifies Portuguese flora species.

Methodology

We started by gathering information regarding the use of deep learning for image classification of flora and fauna, and Citizen Science on smartphones, as well as web services. Afterwards, we conducted a study case to test the tools to be used in the project, i.e., Google Auto ML and TensorFlow.

Following that, we used a similar workflow for the dissertation project. We started by gathering the datasets necessary and generating the models. With the models ready, we developed the Web application and integrated those same models in it.

In the final stages, we tested the models with different test groups and compared their performance through a series of metrics. The best of those models was then tested against state-of-the-art model. Finally, we present our conclusions regarding the project and describe the future work.

1.2 Dissertation structure

In Chapter 2 we explain some background, revolving around flora and machine learning, necessary to understand the project. In Chapter 3 we describe some of the most known mobile apps whose purpose is automatic identification of species. We take a look at each app interface, as well as their architecture, and compare them. In Chapter 4 we describe a case study, done in a museum, where a model was developed to identify sections of a specific exhibit. In Chapter 5 we portray the development of the project, from generating the models to the implementation of the web service. In Chapter 6, we depict the results and analysis of the different tests executed with the models developed. We also display a comparison between a market established model and our best model. Finally, in Chapter 7, we expose our conclusions and detail future work.

Chapter 2

Background

In this chapter we introduce the background for the thesis proposal, including: a description of standard terminology used to classify species in taxonomic terms (Section 2.1); reference to some biodiversity data repositories of interest (Section 2.2); the use of Gaia app for monitoring biodiversity (Section 2.3); and machine-learning concepts, in particular regarding Convolutional Neural Networks (CNNs) (Section 2.4).

2.1 Taxonomy

Before 1735, there was no defined way of categorizing nature's biodiversity, although some work was published by scientists like John Ray [32] in 1682, where he described around 18000 species of plants, and Joseph Pitton de Tournefort [42] in 1700, where he described around 9000 species. In 1735, a Swedish botanist named Carl Linnaeus, published the first edition of his major work, *"Systema Naturae"* [22], where he introduced the binomial naming system to describe species, along with classes like order and genus. This work is the reason Carl Linneaus is considered by many the father of modern taxonomy.

Taxonomy is the theory and practice of delimiting and classifying different kinds of entities [23, 25]. In biology, the term taxonomy refers to the conception, naming, and classification of every single living organism on earth. This process characterized by dividing the organisms into taxons, or groups of populations which are usually inferred to be phylogenetically related. The taxons form what is called the taxonomic hierarchy [33], illustrated in the diagram (Figure 2.1). As we can see in the diagram, the taxonomic hierarchy is divided into eight taxons. Some of these taxons have subcategories (i.e., subspecies).

In the diagram. The domain taxon (bottom) is the most generic one. Species is the most specific taxon, and the name of each species is a result of a binomial nomenclature system, meaning, it is composited of two names, first is the genus, the second is known as the specific epithet, which distinguishes the species from others in the same genus. Taking as example the case of the domestic cat, its species is *Felis domestica*, *Felis* is the genus, first letter uppercase, and *domestica* is the specific epithet, all lowercase. The species of an organism is also known as its scientific name. All of the names, whether to specify the domain or any of the other taxons, are Latin and always written in italic, just like the example above [33].

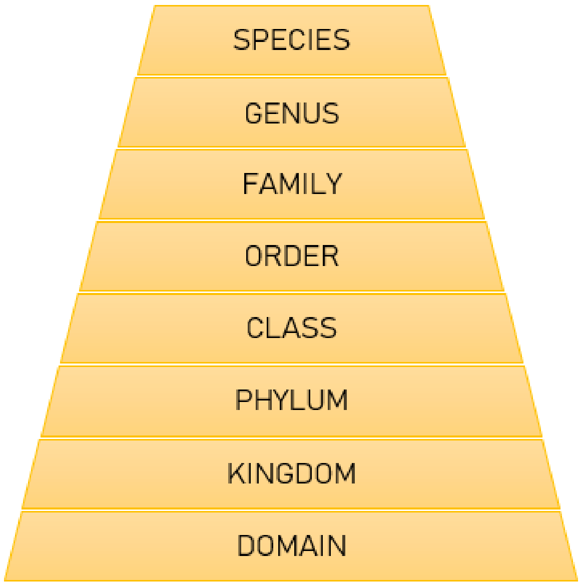
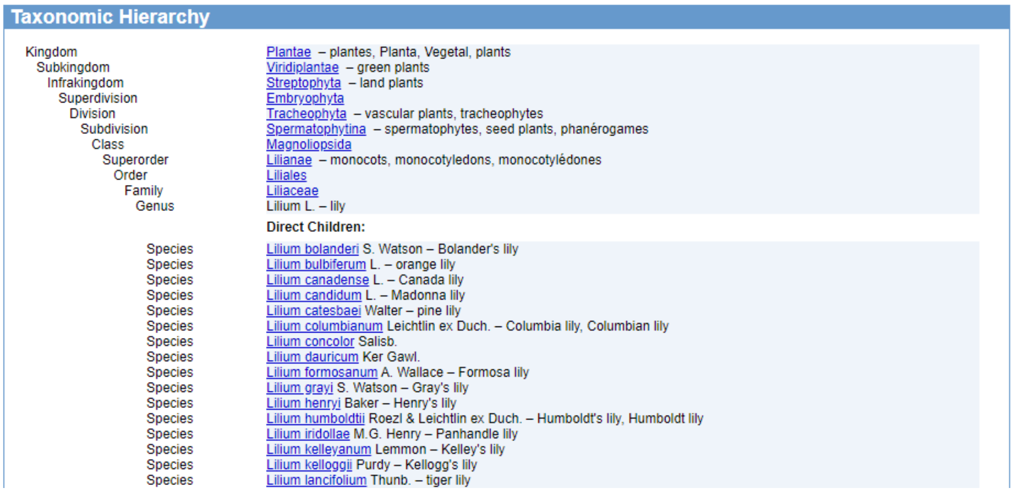


Figure 2.1: Taxonomic Hierarchy

Nowadays, authoritative taxonomic information about a vast range of plants, animals, fungi, and microbes is provided by the Integrated Taxonomic Information System (ITIS), a project that is a consequence of a partnership between USA, Canadian and Mexican agencies, in collaboration with taxonomic experts. In 2013 their database included information on more than one million and three hundred thousand species on a global scale [35], and each of these as a unique Taxonomic Serial Number (TSN) to make sure no organism is checked in more than once. Their website allows costumers to search any of the taxons and provides information about the hierarchy of that same taxon. This process is illustrated in the image below (Figure 2.2) for the genus *Lilium*.



Taxonomic Hierarchy

Kingdom	Plantae – plantes, Planta, Vegetal, plants
Subkingdom	Viridiplantae – green plants
Infrakingdom	Streptophyta – land plants
Superdivision	Embryophyta
Division	Tracheophyta – vascular plants, tracheophytes
Subdivision	Spermatophytina – spermatophytes, seed plants, phanérogames
Class	Magnoliopsida
Superorder	Lilianae – monocots, monocotyledons, monocotylédones
Order	Liliales
Family	Liliaceae
Genus	Lilium L. – lily
	Direct Children:
Species	Lilium bolanderi S. Watson – Bolander's lily
Species	Lilium bulbiferum L. – orange lily
Species	Lilium canadense L. – Canada lily
Species	Lilium candidum L. – Madonna lily
Species	Lilium catesbaei Walter – pine lily
Species	Lilium columbianum Leichtlin ex Duch. – Columbia lily, Columbian lily
Species	Lilium concolor Salisb.
Species	Lilium dauricum Ker Gawl.
Species	Lilium formosanum A. Wallace – Formosa lily
Species	Lilium grayi S. Watson – Gray's lily
Species	Lilium henryi Baker – Henry's lily
Species	Lilium humboldtii Roezl & Leichtlin ex Duch. – Humboldt's lily, Humboldt lily
Species	Lilium iridollae M.G. Henry – Panhandle lily
Species	Lilium kelleyanum Lemmon – Kelley's lily
Species	Lilium kelloggii Purdy – Kellogg's lily
Species	Lilium lancifolium Thunb. – tiger lily

Figure 2.2: ITIS - Information about the genus *Lilium*

We searched a specific taxon, the genus *Lilium* and, as mentioned, all the information about the hierarchy is provided, not only are displayed the names of the species belonging to this genus, but also which taxons the genus *Lilium* is a part of. It is also mentioned that this genus contains thirty-three species and nine subspecies. If we dig a little bit deeper and click on one of the species, i.e. *Lilium candidum*, we obtain information like the common name, the respective TSN, the taxonomic hierarchy and so on.

2.2 Biodiversity data repositories

2.2.1 Global Biodiversity Information Facility

Global Biodiversity Information Facility (GBIF) (Figure 2.3) is a repository that contains data sets from all kinds of species on a global scale. it is purpose is to make biodiversity data and information accessible worldwide, and has around 50000 data sets available.

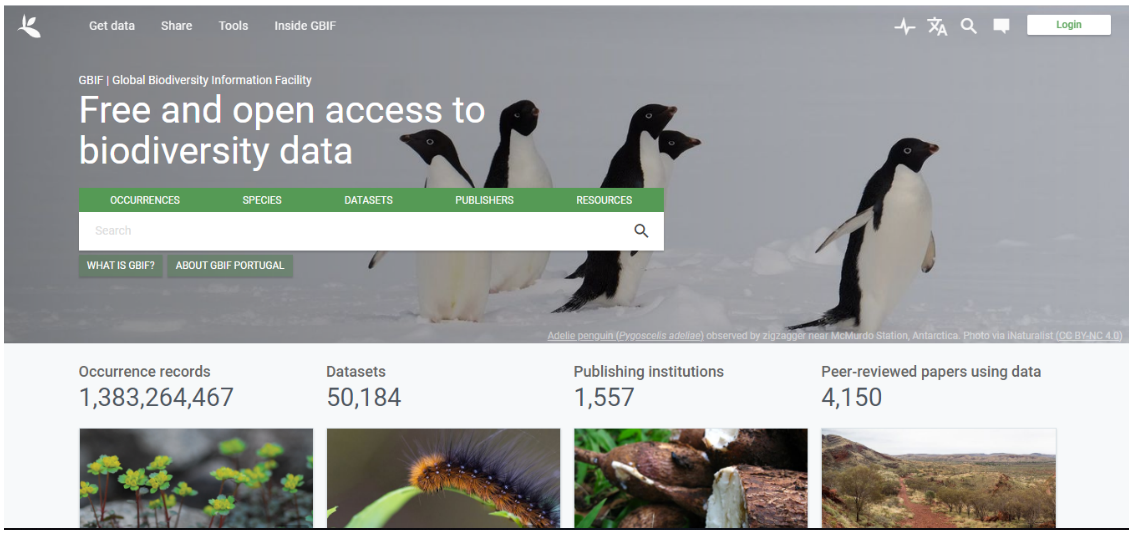

Get data
Share
Tools
Inside GBIF
Login
GBIF | Global Biodiversity Information Facility
Free and open access to biodiversity data
OCCURRENCES
SPECIES
DATASETS
PUBLISHERS
RESOURCES
Search
WHAT IS GBIF?
ABOUT GBIF PORTUGAL
Adelie penguin (Pygoscelis adeliae) observed by zigzagger near McMurdo Station, Antarctica. Photo via iNaturalist (CC BY-NC 4.0)
Occurrence records
1,383,264,467
Datasets
50,184
Publishing institutions
1,557
Peer-reviewed papers using data
4,150

Figure 2.3: Global Biodiversity Information Facility [8].

2.2.2 Flora-On

Flora-On is web platform (Figure 2.4) property of the Sociedade Portguesa de Botânica, that holds information about Portuguese flora. Currently, it has around 450000 observations distributed for 2200 species, this data set is our priority, since it focuses on Portuguese data and it is verified by specialists which makes this data rock solid for training models to identify species.

Figure 2.4: Flora-On [20].

2.3 Machine Learning

Machine Learning (ML) can be perceived as the science of getting computers to learn and act like humans do, and improve their learning over time in an autonomous fashion, by feeding them data and information in the form of observations and real-world interactions [5].

A ML model solves problems in a formal and repeatable way, making judgments about the information that is present in data sets by extracting patterns and acquiring knowledge from the data available and predicting future instances according to a task performed in previous observations [26]. This notion can be adapted to many areas of study, and as a result ML as applications in medical diagnosis [17], genetic analysis [21], computer vision [36], search engines [24], among others [46]. Our area of interest is Computer Vision (CV), where challenges like speech recognition [11] and object detection [4] can be solved with recurrence to a subset of ML, known as Deep Learning (DL).

2.3.1 Neural Networks

Neural Networks (NNs), or Artificial Neural Networks (ANNs), are a set of ML algorithms designed to replicate how actual Biological Neural Networks (BNNs) and their neurons work. Just like humans can recognize an object in an image or a pattern in a data set, these algorithms can be used to perform such tasks and supply information to its users. Below, illustrated in Figure 2.5, is the basic architecture of a NN. It is composed of an input layer, which takes an input, a variable number of hidden layers, which are responsible for processing data and extracting features, and an output layer, to classify the data. Each layer consists of a set of units called neurons, which communicate with each other through signals. These neurons' capabilities come down to combining input signals with an activation function to produce output signals, also known as weights, which are then sent to neurons on the next layer to repeat the process.

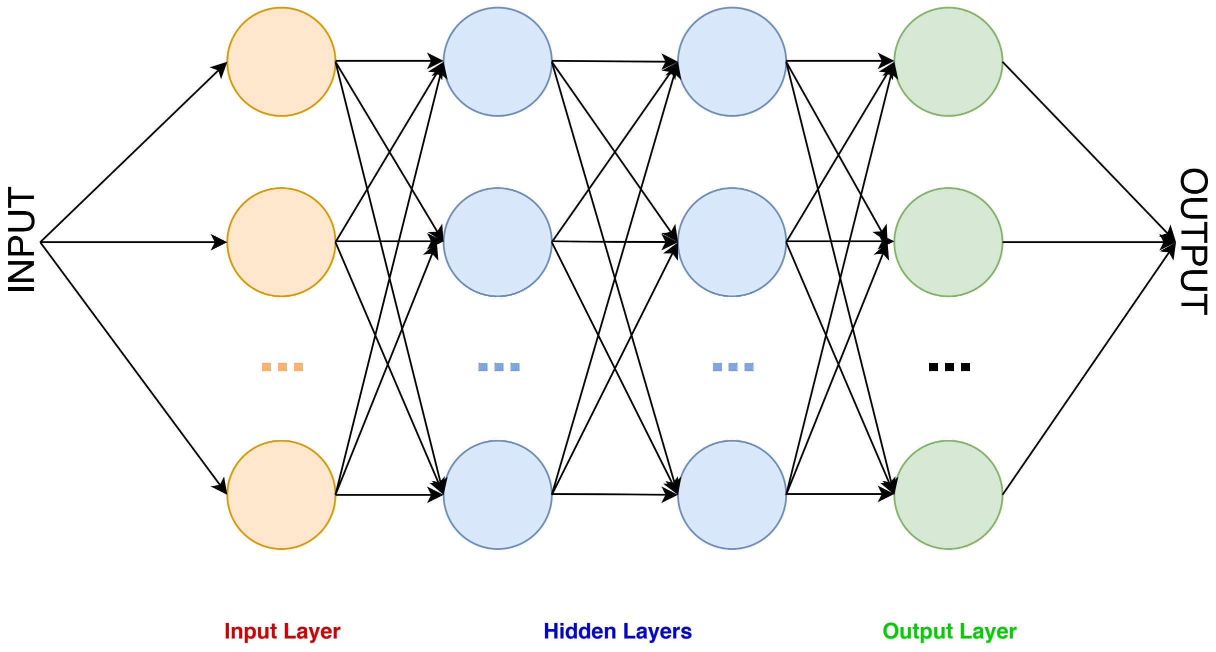


Figure 2.5: Architecture of a Neural Network

Now, to recognize an object in an image a person needs to have seen such object before, or in the case of recognizing a pattern in a data set, it is also necessary some experience to perform such task. These algorithms are no different, they need to be trained to perform a specific task.

2.3.1.1 Training

In order to train a network, we need to provide labeled data so the network can compare its predictions with the correct output and learn from it. Initially, the pixels of an image are fed to the input layer, in which the number of neurons is equal to the number of pixels in the image, following this, weights are assigned to the connections between these neurons and the ones on the next layer, as defined before training.

The input is multiplied by the weight of the connection and the sum is sent to the neurons in the hidden layer as input. Each of the neurons in the hidden layer as a value assigned called bias, this value is added to the input sum and the result is passed through a function known as activation function, that determines which neurons will be activated, that is, which neurons will send information to the neurons in the next layer. This process, illustrated in Figure 2.6, is repeated all the way to the output layer and it is referred to as forward propagation.

Once the output layer is reached, the prediction of the network is based on the neuron of the output layer that has the highest value, these values are percentages and show how confident is the prediction. This prediction is then compared to the actual result (i.e. labels we provide to the network for training), and the network calculates the error based on this comparison, 2.7.

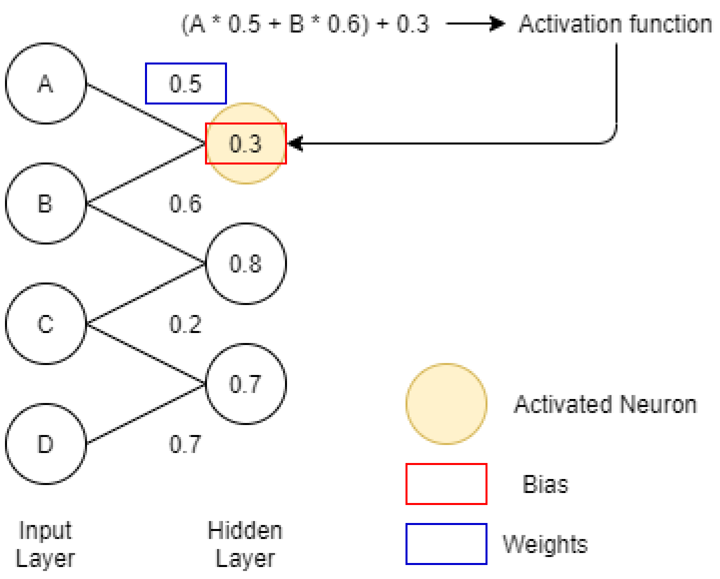


Figure 2.6: Training of a Neural Network

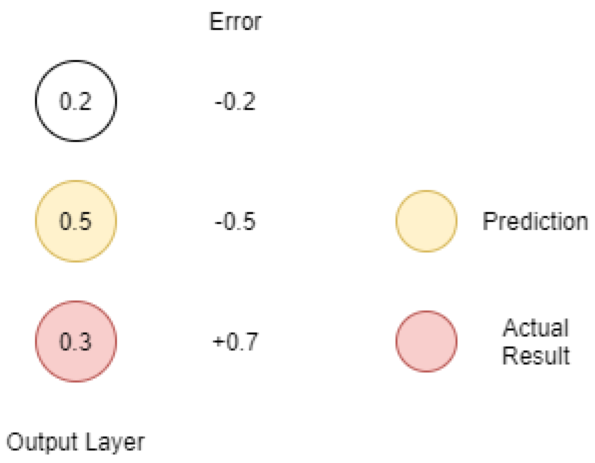


Figure 2.7: Prediction of a Neural Network

The error indicates the distance between the prediction and the actual result, in other words, the actual result should have value 1, and the other two should have 0, hence the error values present in Figure 2.7, needless to say, the model's prediction was wrong. Now, to learn from these mistakes, the error values are sent backwards through the network to adjust the weights of the connections, this process is known as backpropagation. This cycle, forward propagation - backpropagation, is iteratively repeated with all the inputs from a data set until the weights' values are such that the network can, correctly, predict the output in most of the cases.

2.3.1.2 Overfitting and Underfitting

Whether it is before, during or after the training of a network, there are two problems to account for, overfitting and underfitting. The first happens when the data supplied to train the network is very similar, this results in results very uniform which causes the network to make bad predictions with inputs that differ from the data used for training, so a solution is to select a data set for training that has a variety of angles or environments to help the network predict better. The second happens when the data used for training the network is not enough, that way, the network will not capture the patterns of the data, and this will also cause the network to predict successfully at a lower rate, networks need vasts amounts of data to capture features and patterns.

2.3.2 Deep Learning

Deep Learning (DL) has evolved hand in hand with the digital era, which has brought about an explosion of data in all forms and from every region of the world, simply known as big data. This happened because DL models need vasts amounts of training data to be successful.

Conventional ML techniques were limited in their ability to process natural raw data, they needed this data to be pre-processed so that later patterns could be detected or classified. This pre-processing required careful engineering and considerable domain expertise to design a feature extractor that transformed the raw data, such as the pixel values of an image, into a suitable internal representation or feature vector [19]. DL models allow a machine to be fed raw data and automatically discover the representations needed for detection or classification [19], which are known as Deep Neural Networks (DNNs), and are based on ANNs, or simply NNs [38].

2.3.3 Convolutional Neural Networks

CNNs are a DL model and can be split into two main sections, feature extraction, that is gathering features from the input, and classification, classify the input according to its features. For feature extraction it uses a series of convolution layers, which are followed by Fully Connected Layers (Fully Connected Layers (FCLs)), these layers compose the classification section, as we can see in Figure 2.8.

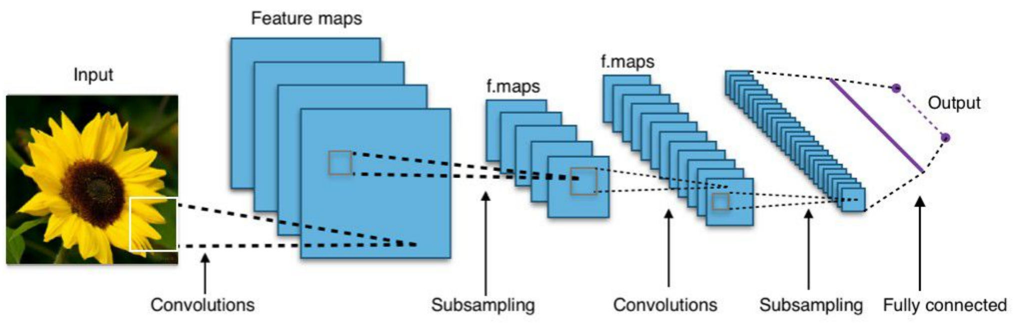


Figure 2.8: Typical CNN architecture. Adapted from [45].

Convolutional Layers

Convolution is a mathematical operation applied to the input using a filter, also known as kernel, as illustrated in Figure 2.9. The kernel slides over the input image one pixel at a time, starting from the top left, it multiplies its values with the overlapping values of the image while sliding over it and adds all of them up to output a single value for each overlap until the entire image is traversed. Convolution is the main characteristic of a CNN, this operation facilitates learning because reduces the size of the output, using a filter, while keeping the main features.

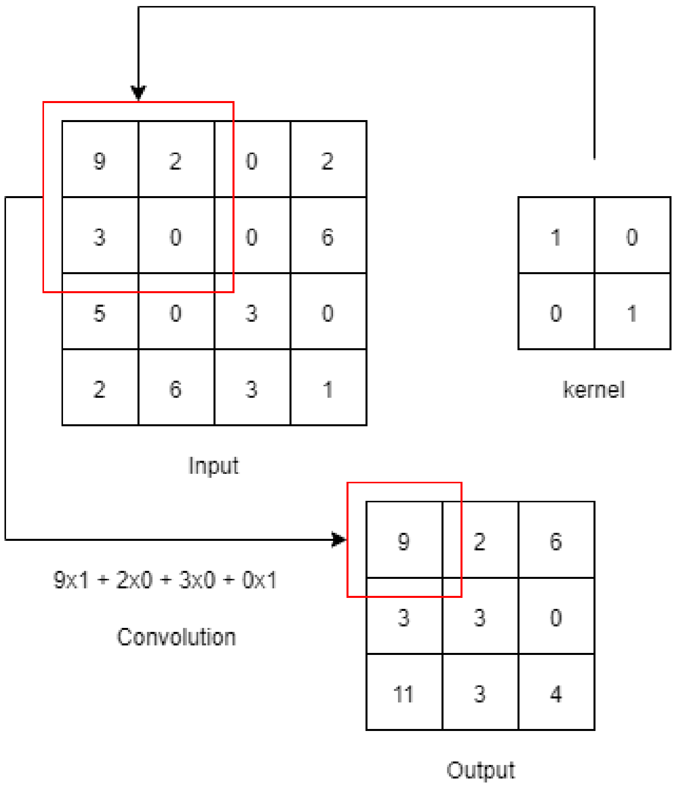


Figure 2.9: Example of convolution.

Activation functions

The purpose of these functions is to help decide which neurons need to be activated. They usually appear in the middle or end of a CNN, right after a convolutional layer. The most common activation function is the Rectified Linear Unit (ReLU), which simply converts all of the negative values to zero and keeps the positive values the same, as shown in Figure 2.10. Therefore, neurons with negative values are not activated.

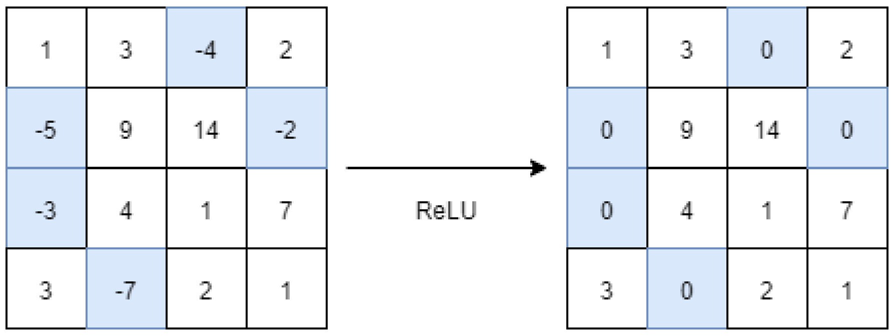


Figure 2.10: ReLU function.

Pooling layers

There are two kinds of pooling, average pooling and max pooling. Average pooling returns the average of all the values from the portion of the image covered by the kernel, while the max pooling selects the maximum number of the same values. In Figure 2.11 is illustrated an example of max pooling. The pooling operation improves statistical efficiency and reduces the size of the inputs for the next layer, this process is called downsampling or subsampling (Figure 2.8).

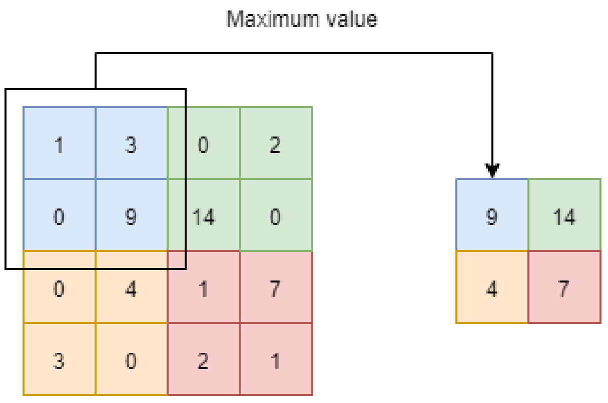


Figure 2.11: Max pooling, using a 2 by 2 filter.

Fully Connected Layers

Once all the data has gone through the convolutional layers, it is to be used by FCL to predict a label for the input. The result of the convolutions is first flattened into a single vector of values. In these layers, all the neurons in a layer are connected to all the neurons in the next layer, just like a usual NN, which means weights and bias are assigned to connections and neurons respectively, to determine the label that best fits the input. Forward propagation and backpropagation are used to adjust these values so the output is the correct one.

Softmax function

The outputs of the last layer of a CNN are not in the probabilities format. Softmax is an activation function used in the end of a CNN to turn the output values into probabilities. This function gathers the output values into a vector of size n, and normalizes it into a probability distribution of n probabilities. The sum of the probabilities must always be 1. Each provability is assigned to a label.

2.3.4 Transfer Learning

As we know, neural networks require large amounts of data to learn from. Learning, which is accomplished by training, can take many hours, or even days. A valuable solution to this problem is the concept of transfer learning, that is, the use of pre-trained models of neural networks that already have information about the general characteristics of a wide range of classes, related or not to the ones we want the network to identify. This allows to take advantage of such information and save considerably amounts of time by training just a fraction of the network, depending on the part of the network that is going to be trained different techniques may be applied. Considering the architecture of CNNs, feature extraction and fine-tuning are commonly used to apply transfer learning. Feature extraction is when only the layers in the classification section are trained from scratch. Since these layers lead to classification, they are trained to classify the new data. In the case of fine-tuning, besides training the layers present in the classification section, some of the top convolutional layers, which contain the most specific features of the data, are also trained to "fine-tune" the parameters that lead to classification.

Chapter 3

State of the Art

In this chapter are described some solutions regarding image classification in the environmental quest area, mainly solutions that are associated with portability, i.e. mobile applications. In the end, is depicted a discussion about the characteristics of these solutions.

3.1 Comparable Solutions

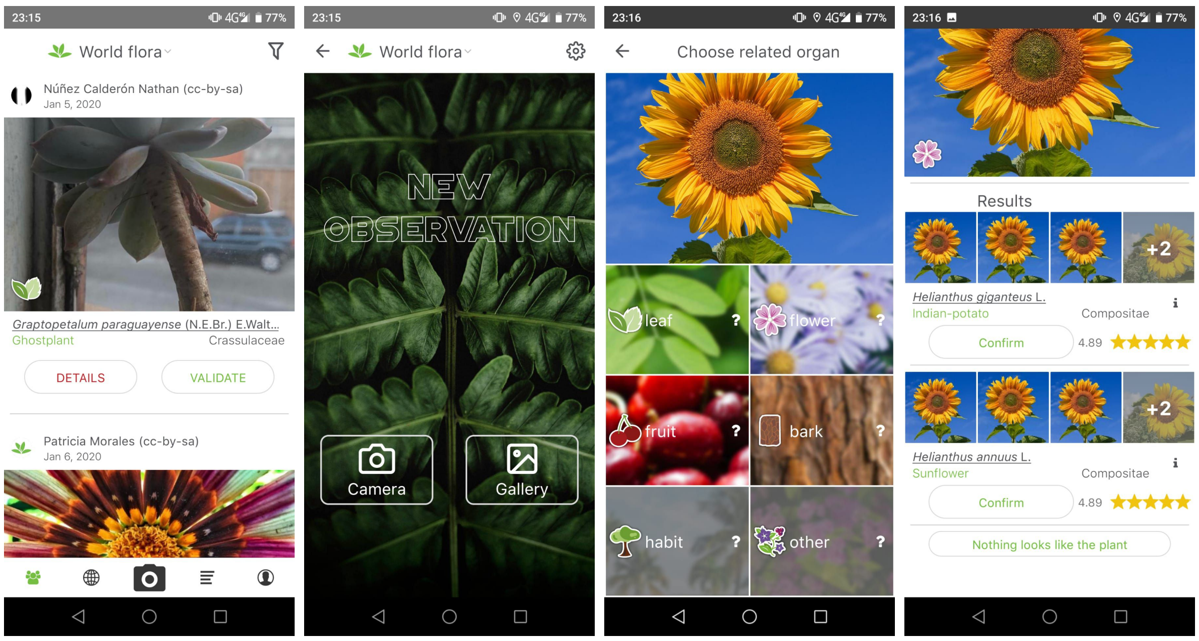
In this section are portrayed and compared different solutions that apply the concept of image recognition, more specifically, mobile apps, which are the focal point of this dissertation. All of them are related to the environment and with a substantial variety of organisms, animals, plants and even fungi. Even though this dissertation focuses on plants, particularly the ones located in Porto district, it is also important to consider mobile applications with a more vast range of organisms, since every used technique should be analyzed and compared to make a solid and fundamental decision about the application's architecture to develop.

3.1.1 PlantNet

PlantNet is a large-scale participatory platform and information system dedicated to the production of botanical data through image-based plant identification [29]. The application is translated in eleven languages and, in 2017, it had been downloaded by more than 3 million users in about 170 countries [2]. The application interface is illustrated below in Figure 3.1.

As we can see, Figure 3.1a illustrates the home interface, on top of the image, "World flora" is the name of the database being used, that is, the data is organized in different databases, each according to a location [29], i.e. if the user uploads a picture of a plant that lives in Africa, the application only identifies such plant if the Africa database is selected, note that some databases have species in common. Still in the home interface, it is presented a feed, where recent observations from others users are displayed. To get the app to identify a plant, the upload of a picture is needed, and for that, there is the camera symbol present in the home interface (Figure

3.1a). Once the user clicks there, the interface present in Figure 3.1b appears on the screen, and as we can see, there are the usual two options, upload a photo from the gallery, or take on right away. After the picture is selected, it is necessary to specify what is that the user wants to be identified (Figure 3.1c). Following that, the top results regarding the identification are shown (Figure 3.1d) and the list of candidate species predicted is displayed in decreasing order of confidence and each species is illustrated by the most similar picture of that species, this allows improving the interact identification process by illustrating the candidate species with images consistent with the observed individual plant.



(a) Home (b) Upload (c) Designation (d) Results

Figure 3.1: PlantNet App Interface

In the results (Figure 3.1d) users can share their observation with the community, whether they have identified it or not, which will then be validated by one of three tools available [2]:

- A web platform called IdentiPlante that is hosted by one of the largest networks of amateurs and expert botanists in the world
- A gamified web application called ThePlantGame that is based on a set of data-driven algorithms
- Embedded validation mechanisms in the Android and Web front-ends themselves

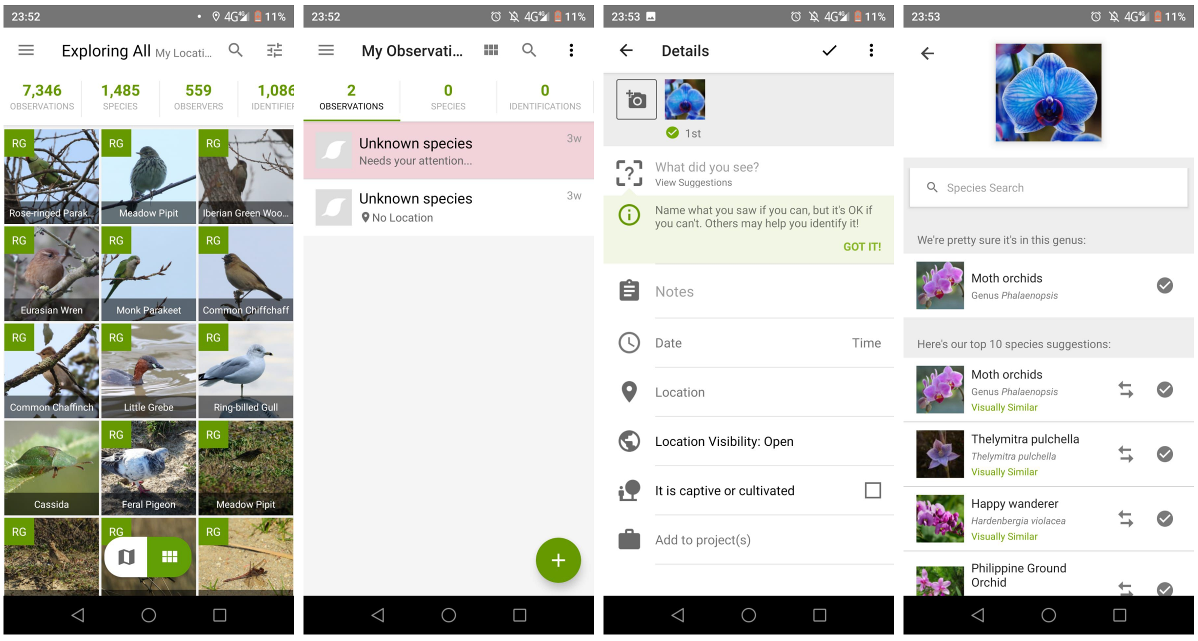
All these validation tools allow the app to cover a growing number of flora and species.

Images are computed by a CNN that is periodically trained, the used CNN architecture is the inception model by Szegedy et al. 2015 [39]. The network is pre-trained with the ImageNet dataset [14] and periodically fine-tuned on PlantNet. The data, in January 2017, was composed of 332 thousand pictures distributed for 10 thousand species. As an observation might be composed of several pictures of the observed plant, the CNN does predictions for every one of them, therefore these predictions need to be fused, so a weighted average of the softmax probability vectors is calculated to supply a final prediction. Note that users may submit a picture of leaves and flowers of the same plant at the same time, in that case, pictures tagged as *flower* are more weighted than pictures tagged as *leaf* because flowers are more discriminant than leaves. The identification of a species can only be done with a connection to the internet, it does not run locally.

3.1.2 iNaturalist and Seek

iNaturalist is a joint initiative by the California Academy of Sciences and the National Geographic Society, it is a platform for sharing information about all kinds of different species. It has a website [15] and a citizen science mobile application (Figure 3.2) that allows users to contribute with observations to help specialists monitor biodiversity across the globe. This application is to record observations of users and details of said observations, such as the location. Seek is a mobile application also developed by iNaturalist because they wanted to create a kid-friendly app that the kids could use without exposing details of the observations such as their exact location, which is illegal without the consent of the parents, in some countries. This said, the app is only for automatic identification of species, it is not a citizen science app.

3.1.2.1 iNaturalist Interface

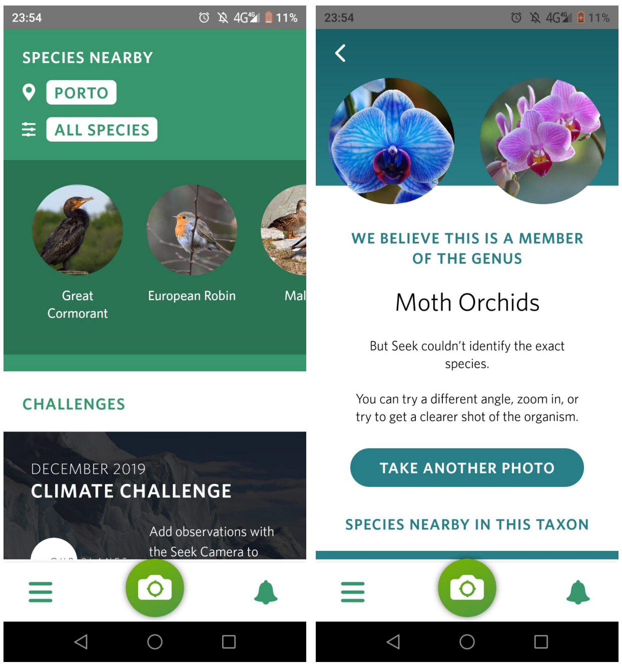


(a) Home (b) Observations list (c) Observation Details (d) Suggestions

Figure 3.2: iNaturalist App Interface

On the home screen, the user can check its observations (Figure 3.2b) through the side menu, once there, it is possible to edit any observation that is registered already or create a new one by clicking on the plus represented in the lower right corner of the screen. By creating a new observation, a few details need to be filled (Figure 3.2c) and one of them is the species. Just like PlantNet, it is possible to add more than one picture of the species, and in the details screen, there is a box that says "What did you see?", which if we click it, supplies possible names for the species to help the user (Figure 3.2d). As said before, this app is a citizen science app to register observations, but it uses models of image classification to provide suggestions for possible name species.

3.1.2.2 Seek Interface



But Seek couldn't identify the exact species.

You can try a different angle, zoom in, or try to get a clearer shot of the organism.

(a) Home (b) Results

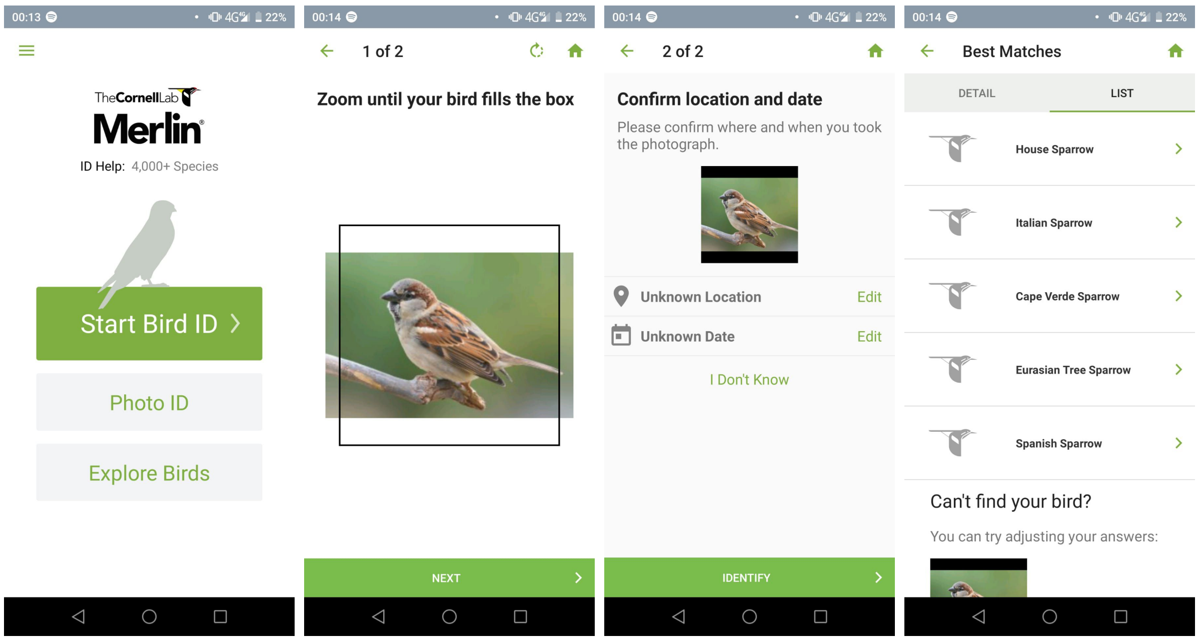
Figure 3.3: Seek App Interface

As mentioned before, Seek is a straight-up automatic identification of species kind of app, to identify a species the user just needs to click on the camera icon present in the home interface (Figure 3.3a) and as soon as the user provides a picture, whether, from the camera or the local memory, the app returns the name of the species present in the picture (Figure 3.3b).

These apps use a dataset that contains data from the iNaturalist website [15], all of the images present in this same dataset were verified by multiple citizen scientist [43], it is composed of 675000 images of over 5000 thousand species. This dataset is different than others in the sense that the pictures are not distributed uniformly throughout the various species, that is, as some species are more likely to be observed than others, this discrepancy between the number of images per species helps the CNN model to predict with better precision [43] since common species are more likely to be observed than species that are rare to be seen on a day-to-day routine, for example, the dataset has around 150000 pictures of plants and only around 5000 of fungi, this because we are way more likely to see a plant than fungi. Within these classes, the same happens. The CNN model used with this dataset is Inception ResNetv2 SE [12].

3.1.3 Merlin Bird ID

Merlin Bird ID is a mobile application to identify birds from all over the world, developed by The Cornell Lab of Ornithology. Dedicated to service and impact, the Cornell Lab is an expert and trusted voice and partner for communities around the globe working together for conservation. Founded in 1915 by Arthur A. Allen, their mission is to interpret and conserve the earth's biological diversity through research, education and citizen science focused on birds [3].



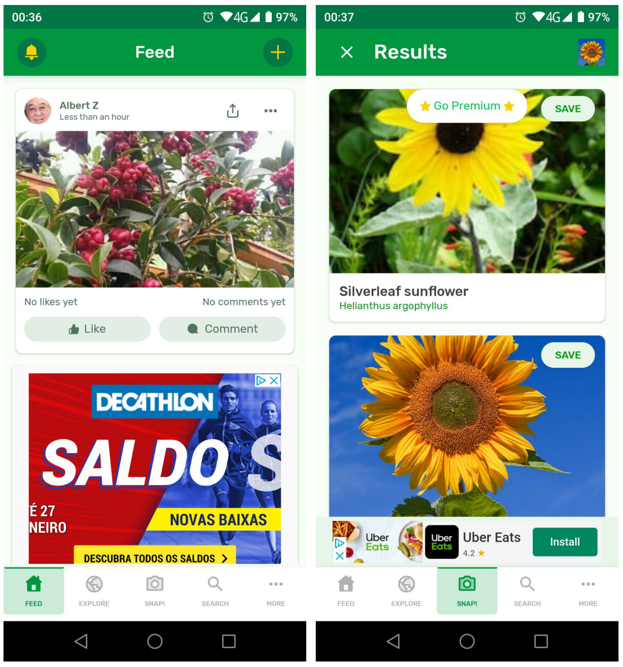
(a) Home (b) Uploaded Picture (c) Details (d) Best Matches

Figure 3.4: Merlin Bird ID App Interface

The interface is pretty simple, to identify a bird species users click on the "Photo ID" button present on the home screen (Figure 3.4a). Next, the user selects a picture, from the local memory or takes on using the camera, and arranges it to fit the bird in the middle of the screen (Figure 3.4b) so the precision of the prediction can be more accurate. Further, the date and location of the observation can be described (Figure 3.4c) although both of them are optional, hereupon the picture is submitted to be analyzed. In Figure 3.4d are displayed the best matches, users can see the list of results and the details of each of them.

3.1.4 PlantSnap

PlantSnap was founded in 2016, it emerged from a simple idea from the founder, he came face-to-face with a plant whose identification he wondered about. The app was created as a way to connect people to the natural world, their database can identify over 600 thousand different species of plants and trees all over the world, with a precision of 90% [31].



(a) Feed (b) Results

Figure 3.5: PlantSnap App Interface

The app uses technologies as auto-detect and augmented reality to help users taking a photo that is better for the app to identify the plant in question. It has a feed, where users can share their observations and see other users' observations as well (Figure 3.5a) a picture is uploaded using the camera or the gallery, as usual, by clicking in the "snap" icon on the bottom of the screen (Figure 3.5). The results have the format illustrated in Figure 3.5b. With 30 million downloads and more than 125 million pictures identified, it is one of the most used apps for identifying plants' species [31].

3.2 Run-through

In terms of interface, the apps are all similar, the user uploads/takes a picture and the results are displayed. A particular case about PlantNet, the user needs to specify what is in the picture, meaning, a leaf, flower, fruit, bark, habit, or other. About the observation, some information can be registered, in the case of Merlin Bird ID, the location and date are optional for the user to specify, in the case of iNaturalist, considering this is primarily a citizen science app, some details can also be specified besides the ones mentioned before, such as location visibility, it can be open, obscured or private, and it is possible to register if the plant is captive or cultivated. An interesting feature about PlantNet and PlanSnap, both present a social feed where users can share and see each other's' observations.

These apps have different areas of study, Merlin Bird ID is the only app mentioned specialized in birds, it identifies over 4500 bird species from North and South America, Europe, Asia, and Australia. iNaturalist and Seek are the ones with the widest range, they identify all kinds of species, whether it is plants, birds, reptiles, fungi, and so on, from all over the world. PlantNet and PlantSnap identify plant species only, although PlantSnap includes trees too, also from everywhere.

One important aspect we noticed, only Merlin Bird ID and Seek can identify species locally, meaning, they don't need the connection of the internet to process and analyze the pictures, this probably because, as Merlin only focuses on birds their dataset has about 4500 species, and Seek has about 14000 species, these numbers are small compared to apps like PlantSnap that has 600 thousand species that compose their dataset. While with small numbers the computation can be executed in the mobile device and still give good results within proper time, apps with datasets somewhat extensive, e.g. PlantSnap, need to be connected to the servers so the computation can be executed there, because it is way more complex than other examples, and if executed locally would take a lot of time and it would not work on some mobile phones.

Below, the table 3.1, illustrates the essential characteristics of each of the analyzed mobile applications.

Table 3.1: Apps comparison

App / Attributes	Local run	Dataset Size (number of species appr.)	Kingdoms	Range	CNN model
PlantNet	No	21000	Plantae	Global	Szegedy et al. 2015
iNaturalist	No	14000	Plantae, Animalia, Fungi and Protozoans	Global	Inception ResNetv2 SE
Seek	Yes	14000	Plantae, Animalia, Fungi and Protozoans	Global	Inception ResNetv2 SE
Merlin Bird ID	Yes	4500	Animalia(Birds)	North and South America, Europe, Asia and Australia	?
PlantSnap	No	600000	Plantae	Global	?

Chapter 4

Case Study

This chapter describes a brief case study we made at a museum venue that illustrates in a simple manner how Google Auto ML Vision and TensorFlow can be used for deriving and deploying DL models for image classification. The aim was to derive a simple DL model to identify museum sections that could be later be deployed to classify live images obtained using a smartphone. We describe the museum venue (Section 4.1), then the process of deriving (4.2) and deploying (4.3) the DL model.

4.1 Galeria da Biodiversidade da Universidade do Porto

Galeria da Biodiversidade da Universidade do Porto (GBUP) is a place in which the arts connect with biology and natural history, fostering a wide range of sensorial experiences, crafted to celebrate the diversity of life. Herein, visitors will encounter a set of exhibitions, one of which is permanent and organized according to 15 major topics covering major aspects of the biological and cultural diversity we now know. The permanent exhibit at GBUP, illustrated in the photos of Figure 4.1, provides an amazingly rich and diversified set of museographic resources, ranging from mechanical models to sophisticated multimedia and audiovisual installations.

To get familiarised with the practical aspects of model derivation and deployment, we decided to perform a simple case study focused on the permanent exhibition at GBUP. The other goal was to make a preliminary evaluation of if and how DL models could be used in conjunction with smartphone apps for indoor location, the subject of another ongoing project.

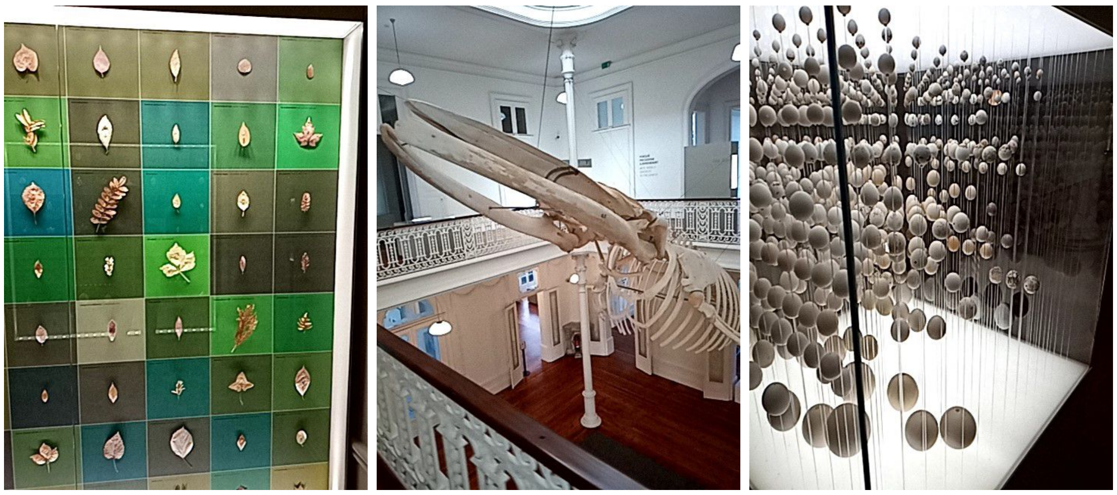

(a) Folhas (b) Baleia (c) Caixa de ovos

Figure 4.1: Photos from different sections of the GBUP exhibit.

4.2 Model derivation

Google AutoML Vision (AMLV) is a cloud service to perform automated training of a deep neural network and obtain the resulting DL model in several formats [10]. Figure 4.2 illustrates the necessary steps necessary to generate a model using AMLV. In brief, AMLV requires a dataset of images and their classification labels to perform automatic training of a CNN. After training, the resulting model can be obtained in several formats including TensorFlow Lite, the most adequate form of model deployment for web-based or Android smartphone apps.

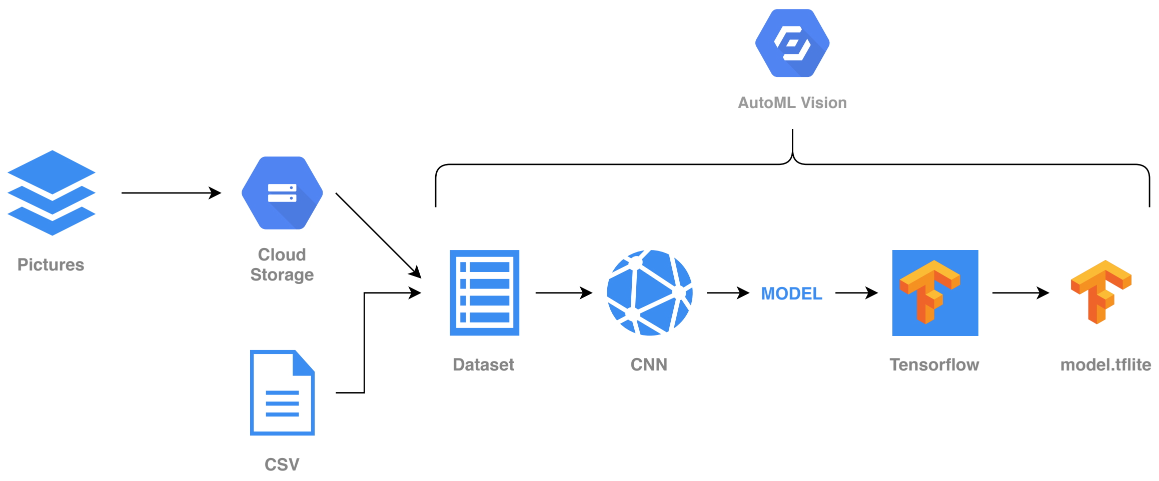


Figure 4.2: Workflow for deriving the GBUP model.

To derive the model, we started by taking photos of different sections of the exhibit, such as those shown back in Figure 4.1, and placed them in a bucket in the Google Cloud Storage [9]

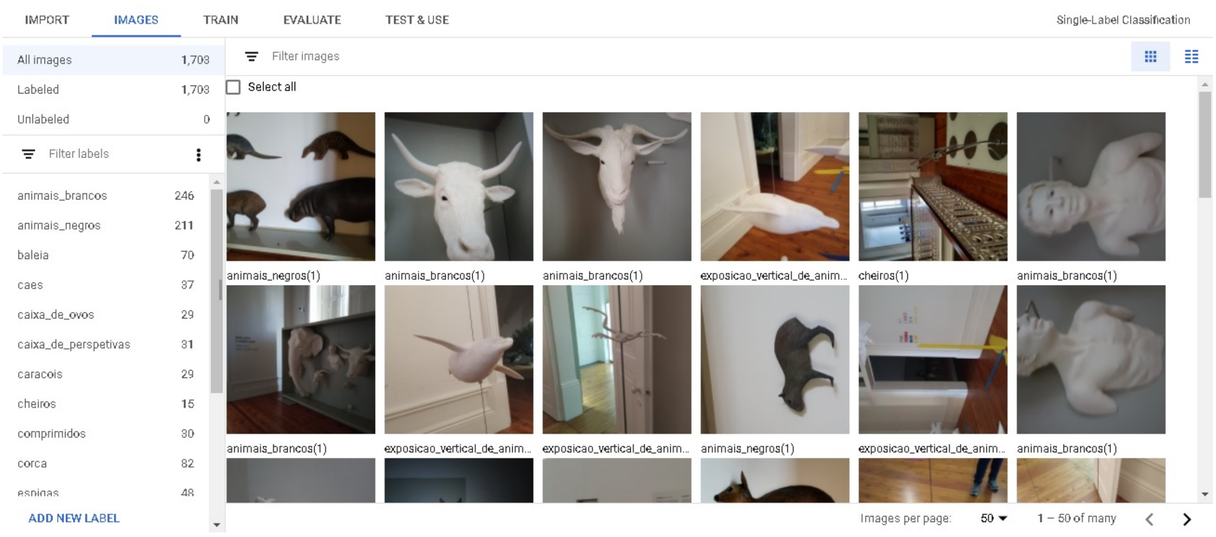
(GCS), a pre-requisite by AMLV to store image files. We then prepared a CSV file with the image labels for each picture that should be used to train the model. The CSV file is composed of path-label pairs, where a path refers to the GCS path for an image and a label classifies the image in a certain category, as follows:

```
Location,Label
gcs://path/to/image.png,Baleia
...
```

Table 4.1 lists the classification labels and the corresponding number of images in the resulting dataset. Once AMLV processes the CSV file and the images, it displays the dataset as shown in Figure 4.3, which can now be used train a CNN.

Table 4.1: Number of images per section of the GBUP exhibit.

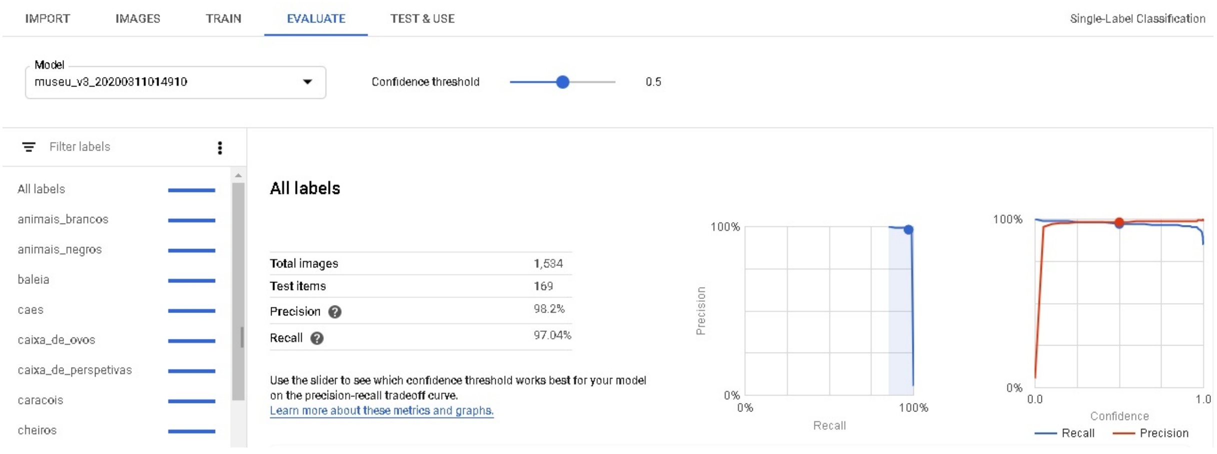
Section	# Images	Section	# Images
animais brancos	246	corça	82
animais negros	211	espigas	48
baleia	70	exposição vertival de animais	577
caes	37	folhas	70
caixa de ovos	29	ovos esféricos e ovóides	13
caixa de perspetivas	31	pavão	31
caracois	29	sementes	33
cheiros	15	veado	47
comprimidos	30	nula	104



All images	1,703
Labeled	1,703
Unlabeled	0
animais_brancos	246
animais_negros	211
baleia	70
caes	37
caixa_de_ovos	29
caixa_de_perspetivas	31
caracois	29
cheiros	15
comprimidos	30
corca	82
espigas	48

Figure 4.3: Google Auto ML – dataset panel.

AMLV separates the images of each label into three groups, training, validation, and testing randomly sampled from the data set with the following proportions: 80 %, 10% and 10%. For example, in Table 4.1, we have 70 images for the **baleia** label, hence AMLV uses 56 for training, 7 for validation and 7 seven for testing. Once training is done in the cloud, a process that lasts several hours, AMLV provides precision and recall metrics (discussed later in Chapter 6) for the generated model, providing a glimpse of how, potentially, good or bad, the model is. Figure 4.4 illustrates the evaluation panel for the museum model.



Model: museu_v3_20200311014910

Confidence threshold 0.5

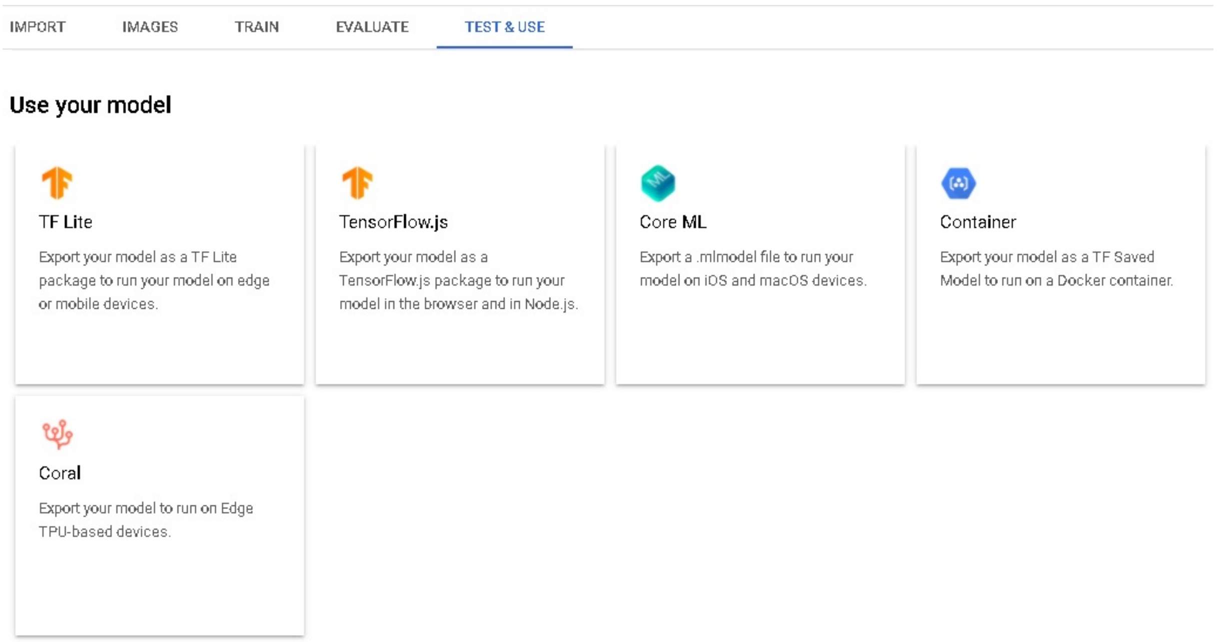
All labels

Total images	1,534
Test items	169
Precision	98.2%
Recall	97.04%

Use the slider to see which confidence threshold works best for your model on the precision-recall tradeoff curve.
Learn more about these metrics and graphs.

Figure 4.4: Google Auto ML – evaluation panel.

4.3 Model deployment



Use your model

TF Lite
Export your model as a TF Lite package to run your model on edge or mobile devices.

TensorFlow.js
Export your model as a TensorFlow.js package to run your model in the browser and in Node.js.

Core ML
Export a .mlmodel file to run your model on iOS and macOS devices.

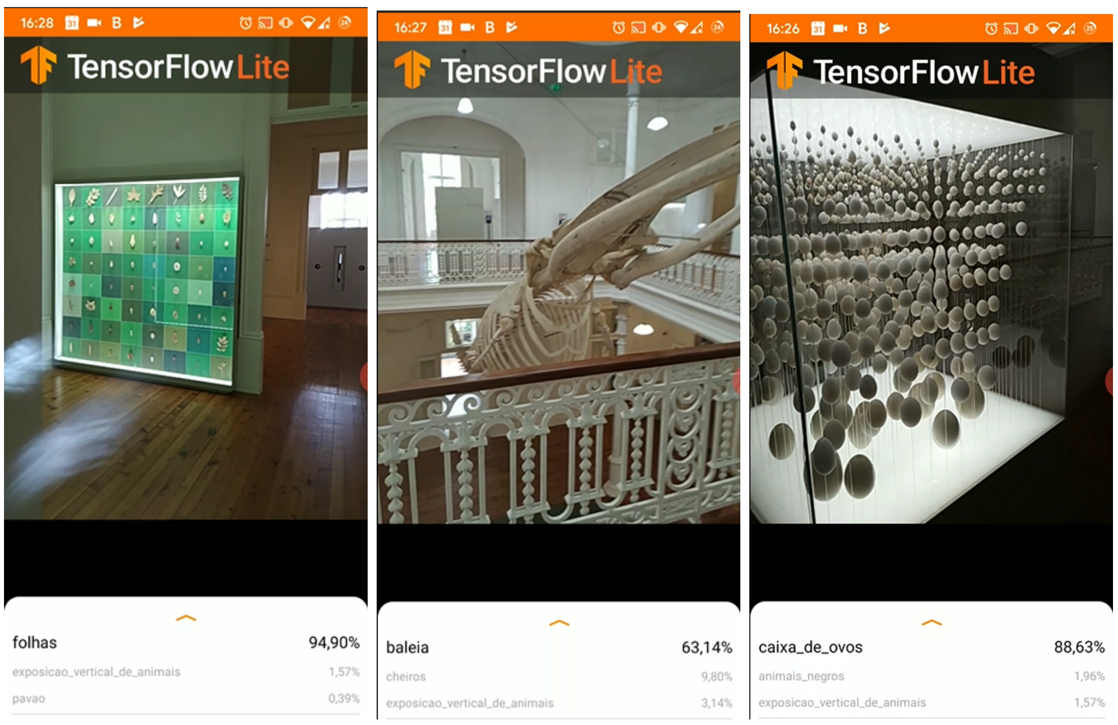
Container
Export your model as a TF Saved Model to run on a Docker container.

Coral
Export your model to run on Edge TPU-based devices.

Figure 4.5: Google Auto ML – export options.

Once a DL model is derived by AMLV, it can be deployed in a number of ways and for a plethora of platforms, including mobile and web applications, illustrated in Figure 4.5. For deploying this model in an Android mobile app, we exported the model as a TensorFlow Lite format [40]. We then integrated the model into a TensorFlow Lite demonstration app for image classification [41] to test it out.

Figure 4.6 provides a few screenshots from the Android app running live at GBUP. The app works by capturing video frames on-the-fly and submitting the frame images to the DL model interpreter for classification. The model returns a confidence level from 0 to 100 per each of the labels, and the app displays the labels with the highest confidence level along with the live video feed. As shown in the screenshots, and by comparison to the photos presented earlier in Figure 4.1, an accurate classification with high confidence could be obtained for some of the museum sections, in spite of the modest size of the dataset we used to derive the model.



folhas 94,90%
exposicao_vertical_de_animais 1,57%
pavao 0,39%

baleia 63,14%
cheiros 9,80%
exposicao_vertical_de_animais 3,14%

caixa_de_ovos 88,63%
animais_negros 1,96%
exposicao_vertical_de_animais 1,57%

(a) Folhas (b) Baleia (c) Caixa de ovos.

Figure 4.6: TensorFlow Lite app screenshots.

Overall, this case study allowed us to get experience with the different tools we later used for flora identification models, described in the following chapters. It also provided a promising preliminary evaluation of the use of DL models in conjunction with smartphone apps for indoor location.

Chapter 5

Development

This chapter presents the different steps for building DL models for flora identification comprising the data collection workflow, the model derivation via AMLV, and the model integration in a Web application that provides classifications for images submitted by users. First, we describe the construction of the DL models (Section 5.1). Next, we describe the DL deployment in the web application and the app implementation (Section 5.2).

5.1 Model development

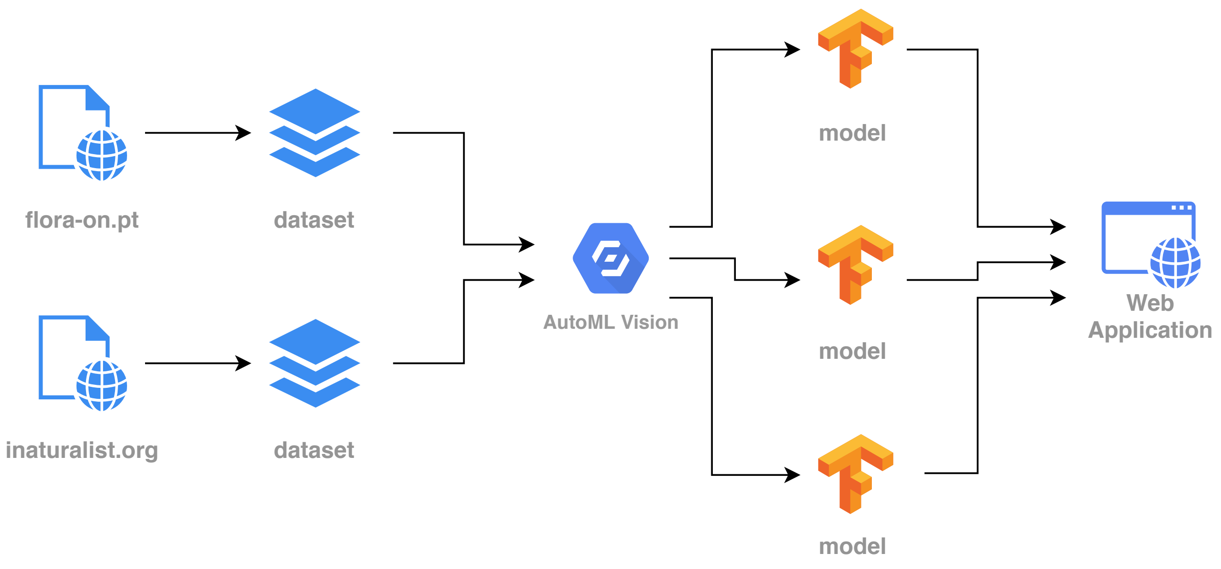


Figure 5.1: Overview of model development.

As discuss in Chapter 2, several data repositories can be used as possible sources of cataloged flora images. From these, we found it convenient to build flora datasets from images found in the Flora-On web portal by Sociedade Portuguesa de Botânica [1], and from the iNaturalist citizen science platform that were tagged as "research-grade" observations [13]. The aim was to derive

DL models using a workflow, illustrated in Figure 5.1, that is similar in nature to the preliminary case study (Chapter 4), but involves image datasets that can have great volume and variety.

Flora-On provides a comprehensive archive of autochthonous Portuguese flora in terms of covered species, even if the number of images available is not vast. Moreover, images consist of photos taken by botanic experts that tend to exhibit distinctive traits for a given species. As for iNaturalist, we could easily obtain a catalog of flora images from iNaturalist through the standardised interface of the GBIF portal, specifying Portugal as the country of interest. Images from iNaturalist tend to have varying quality, cover less species, and are classified by a crowd-sourcing process that may be error-prone when compared to a controlled process by botanic experts, but, on the other hand, a higher volume of images is available. We later also collected iNaturalist images from Spain, trying to cover under-represented species.

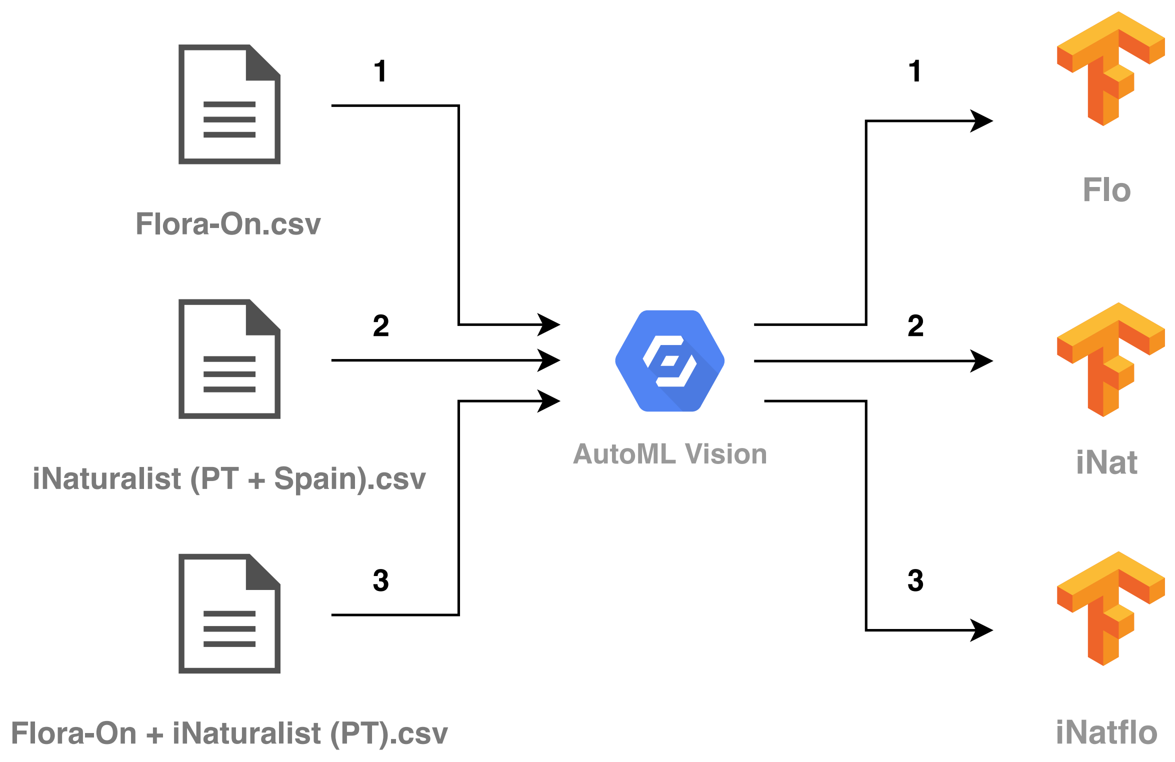


Figure 5.2: Dataset construction and resulting models.

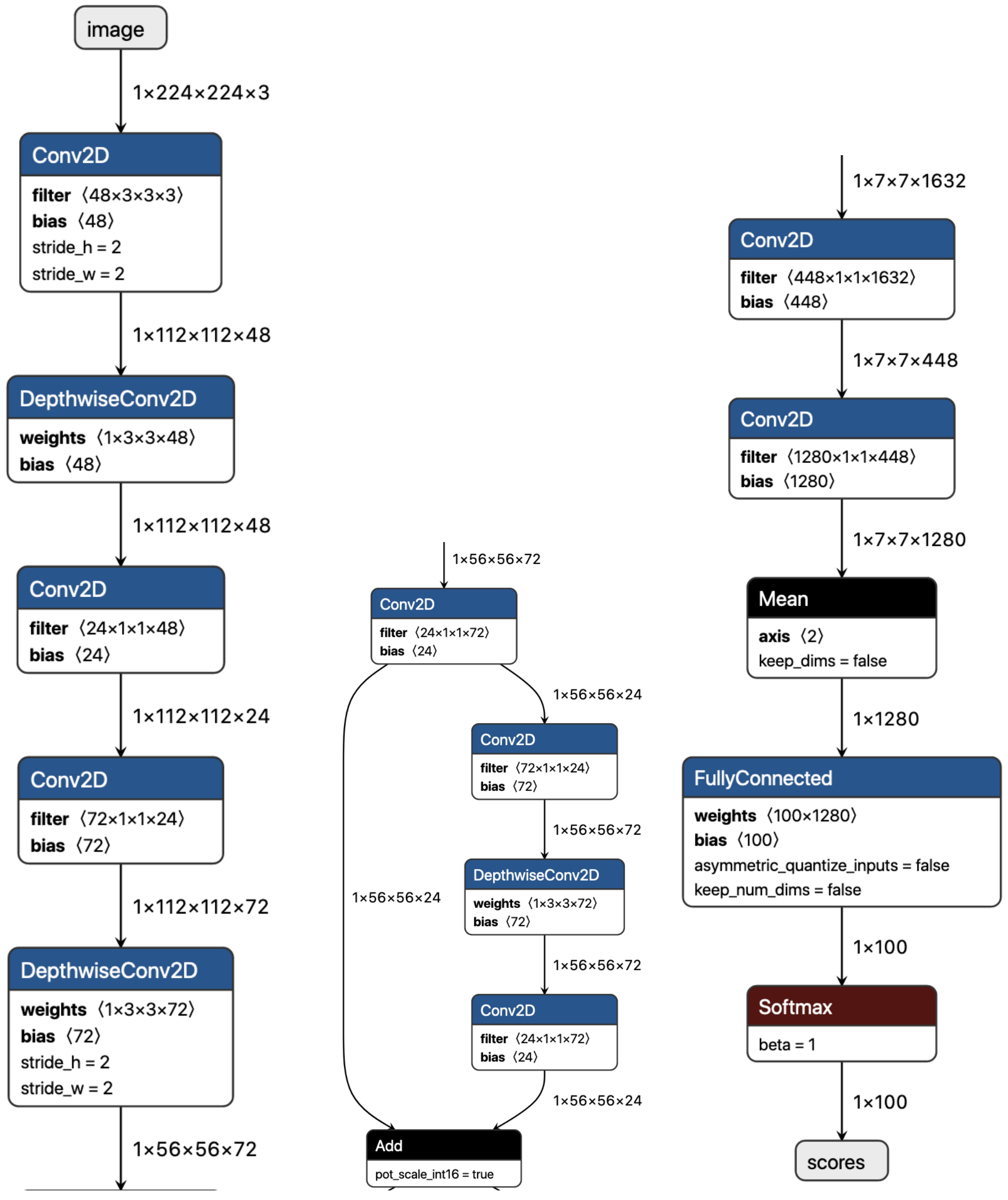
Once the image data was appropriately stored in the cloud for use by AMLV, we played mix-and-match with it by creating three datasets:

1. **Flo**: derived from Flora-On images alone;
2. **iNat**: derived from iNaturalist images, both from Portugal and Spain, and;
3. **iNatFlo**: derived from Flora-On images and iNaturalist images from Portugal, i.e., using Flora-On as supplement to iNaturalist images from Portugal, rather than iNatularist photos from Spain

From these datasets, we derived corresponding DL models, as illustrated in Figure 5.2. The overall process of deriving models using AMLV was the same as described earlier in Chapter 4.

5.1.1 CNN structure

Figure 5.3 illustrates the basic architecture of the (deep) CNN embedded in all the TensorFlowLite models derived using AMLV. The screenshot fragments were obtained using the Netron tool for visualisation of deep learning models [34]. The neural network is 173-layers deep with distinctive top and bottom layer, but a repetitive structure in the middle layers. We discuss these next.



(a) Initial layers. (b) Middle pattern layers. (c) Final layers.

Figure 5.3: CNN structure.

As we can see, through Figure 5.3a, the CNN used starts with a series of convolutional layers. These layers all have different parameters, particularly the filter, also known as kernel. The input is an image with the parameters 224x224x3, those being, height, width, and the number of channels, respectively. An input image usually has 3 channels, RGB. In the five layers, two kinds of convolutions are executed, a normal convolution and a depth-wise one.

In the middle layers (Figure 5.3b), the result of a convolutional layer is used in two fronts and then put together through an *Add* layer. This kind of layer requires a list of tensors as input, and they must have the same shape, in this case, 1x56x56x24. The output also has that same shape. Tensors are the results of the layers.

Regarding the final layers (Figure 5.3c), we have two more convolutional layers and a *Mean* layer. This layer takes a tensor and it does the mean of the integer of one of the tensor's axis. Following this, we have the fully connected layers and the softmax function, just like described in section 2.3.3.

5.2 Web application

We now describe the functionality of the web application developed to process flora images and present classification results to the users. Figure 5.4 illustrates the many steps of that process. The basic idea from the user's perspective is that he/she selects a certain number of images from one species and submits them. Following that, a web page with the results is displayed. The remaining of the section focuses on each step in detail.

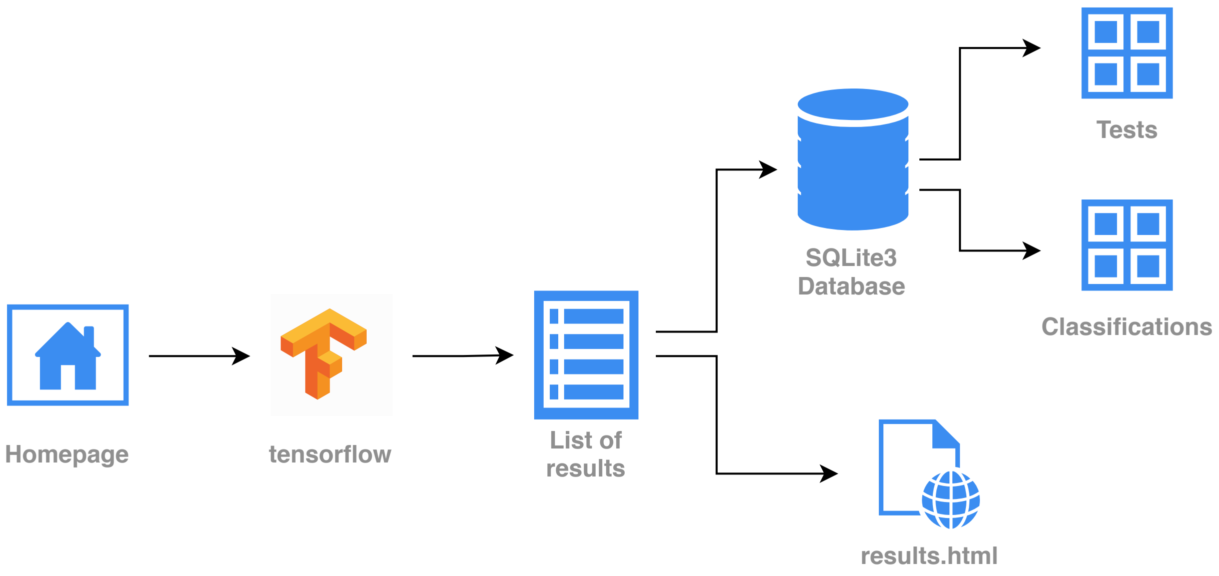


Figure 5.4: Web app scheme.

5.2.1 User interface

The homepage of the app, Figure 5.5, is a simple one. It has three fields, two mandatory, *User* and *Species*, and an optional text area for comments.

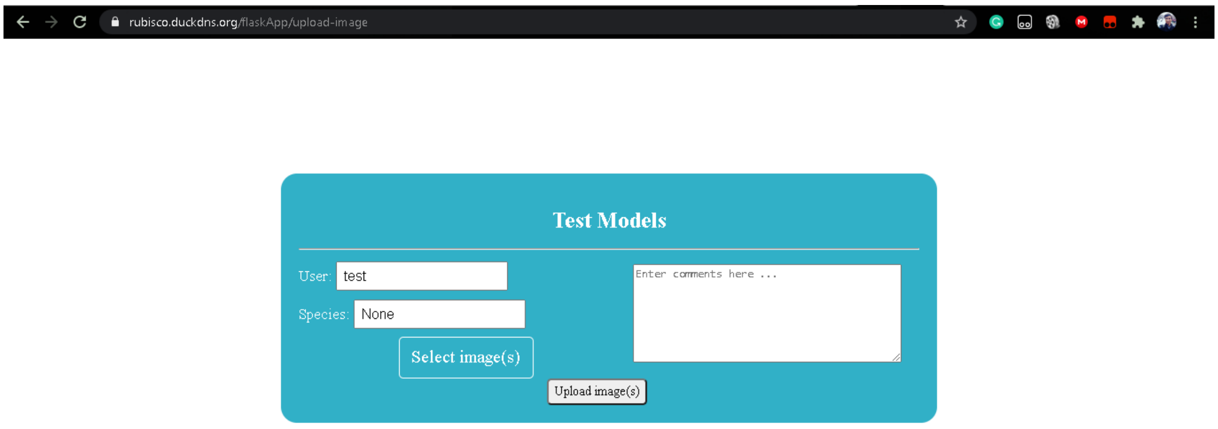
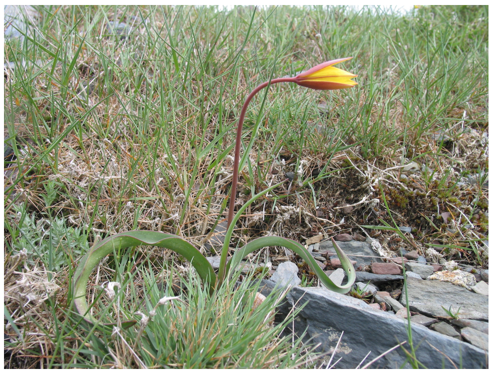


Figure 5.5: Web app homepage.

User, just like the tag suggests, is the name by which the person that submits the images for analyses is identified. *Species* corresponds to the species of the plant present in the images submitted. In the case of the user not knowing or wishing to specify what species is (or are) in the images, the option *None* is available. The images are selected through the button *Select image(s)*. The *comments* area is optional, just in case the user wants to add some note about the images being submitted. The *Upload image(s)* button starts the backend process to obtain the results.

Figures 5.6a and 5.6b are examples of pictures submitted, and both of the same species as mentioned. Results show the image, its name, and the table with the first five classifications assigned by each model (Figure 5.7). Classifications are accompanied by their respective confidence values. The blue table cells represent the classifications that match the one submitted by the user.


(a) Tulipa sylvestris 1.

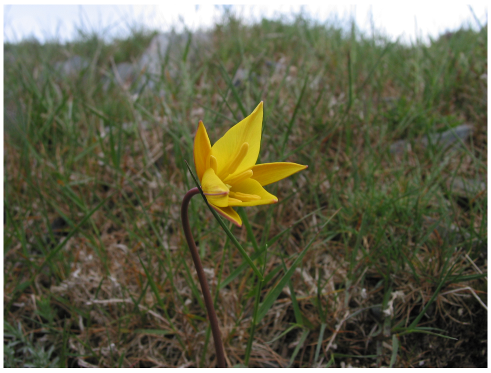
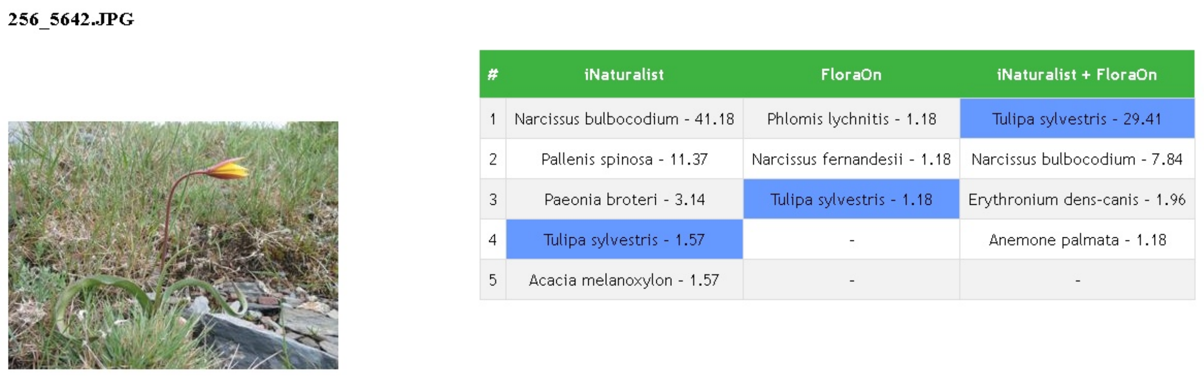
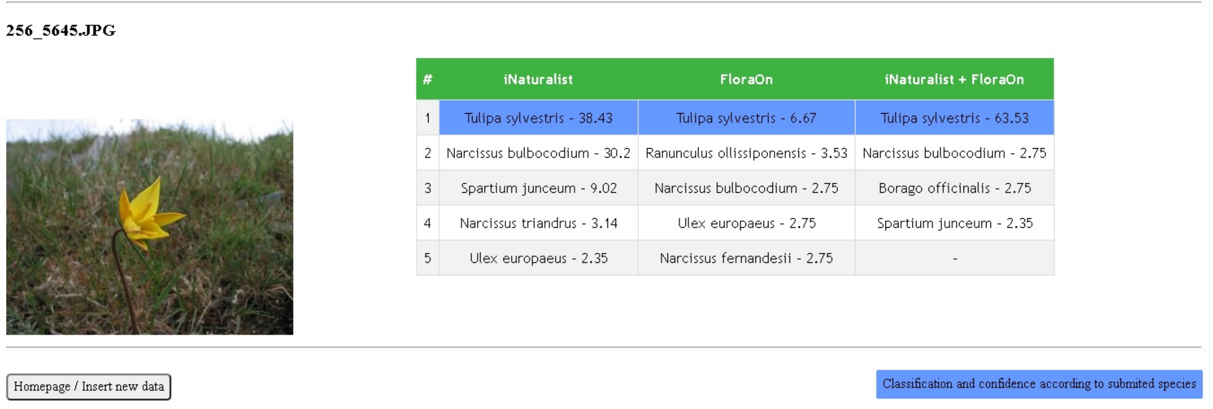

(b) Tulipa sylvestris 2.

Figure 5.6: Some of the images tested.


256_5642.JPG

#	iNaturalist	FloraOn	iNaturalist + FloraOn
1	Narcissus bulbocodium - 41.18	Phlomis lychnitis - 1.18	Tulipa sylvestris - 29.41
2	Pallenis spinosa - 11.37	Narcissus fernandesii - 1.18	Narcissus bulbocodium - 7.84
3	Paeonia broteri - 3.14	Tulipa sylvestris - 1.18	Erythronium dens-canis - 1.96
4	Tulipa sylvestris - 1.57	-	Anemone palmata - 1.18
5	Acacia melanoxylon - 1.57	-	-

(a) Results Figure 5.6a.


256_5645.JPG

#	iNaturalist	FloraOn	iNaturalist + FloraOn
1	Tulipa sylvestris - 38.43	Tulipa sylvestris - 6.67	Tulipa sylvestris - 63.53
2	Narcissus bulbocodium - 30.2	Ranunculus ollissiponensis - 3.53	Narcissus bulbocodium - 2.75
3	Spartium junceum - 9.02	Narcissus bulbocodium - 2.75	Borago officinalis - 2.75
4	Narcissus triandrus - 3.14	Ulex europaeus - 2.75	Spartium junceum - 2.35
5	Ulex europaeus - 2.35	Narcissus fernandesii - 2.75	-

Homepage / Insert new data
Classification and confidence according to submited species

(b) Results Figure 5.6b.

Figure 5.7: Tests results for *Tulipa sylvestris*.

5.2.2 Implementation

This app was developed using HTML, JavaScript and CSS for the frontend, and Python Flask for the backend. Flask is a micro web framework written in Python [6]. Figure 5.8 and Listing 5.1 represent the project structure and the APIs used throughout the process, respectively.

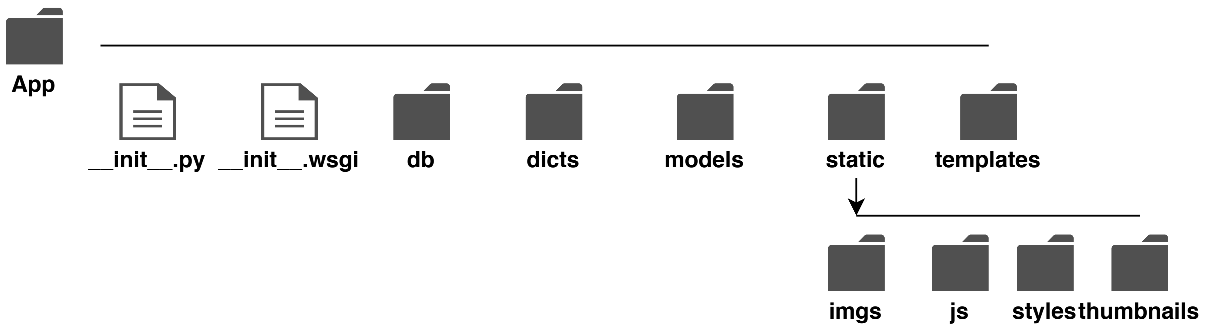


Figure 5.8: Project structure.

```python
from flask import Flask, render_template, url_for, redirect, request, session
import tensorflow as tf
import numpy as np
from PIL import Image
import sqlite3
```

Listing 5.1: Imported APIs.

In summary the main APIs used and corresponding role are:

- `flask`: the overall Python framework to build web applications;
- `tensorflow`: TensorFlow Lite functionaliyy;
- `numpy`: a library for numeric datatypes and mathematical functions;
- `Image`: an image processing library;
- `sqlite3`: access to Sqlite3 databases;

5.2.3 Backend

Once the data is submitted, it is treated in the function `upload_image` (Listing 5.2).

```python
@app.route("/upload-image", methods=["GET", "POST"])
def upload_image():
    if request.method == "POST":
        if request.files:
            images = request.files.getlist("images[]")

        if request.values:
            f\_name = request.form['fname']
            expert\_label = request.form['species\_in']
```

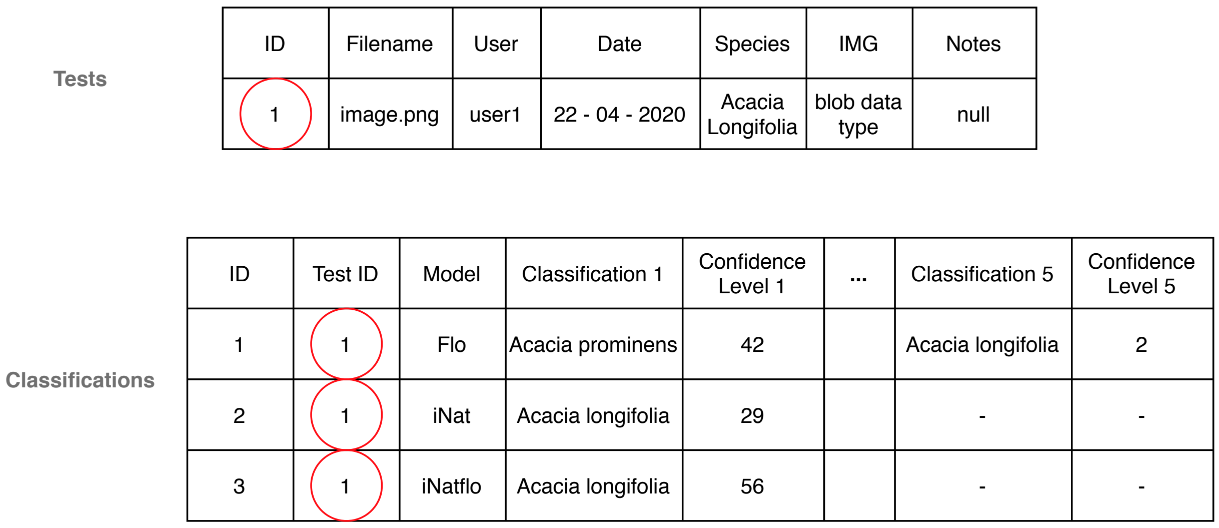
```
10                comments = request.form['comments']
11                if comments == "":
12                    comments = "null"
13            else:
14                print('No values in request!')
15                redirect(request.url)
16
17            extension = True
18
19            for image in images:
20                if not check_extension(image.filename):
21                    print("Error: file extensions allowed %s!" %
                          (ALLOWED_EXTENSIONS))
22                    extension = False
23
24            if extension == False:
25                redirect(request.url)
26            else:
27                names = ""
28                for image in images:
29                    filename = secure_filename(image.filename)
30                    image.save(os.path.join(app.config["IMAGE_UPLOADS"], filename))
31                    names = names + filename + ","
32                    print('Image has been saved!')
33
34                return redirect(url_for('loading_results', names=names,\
35                    f_name=f_name, expert_label=expert_label, comments=comments))
36        else:
37            print('No image selected!')
38            redirect(request.url)
39
40        return render_template("upload.html")
```

Listing 5.2: Function `upload_image`.

In the first part of the function (Listing 5.2) we get the list of filenames for the submitted images (lines 4-5) and gather the information from the *User*, *Species* and comments areas (lines 7-12). In case some mandatory value is missing, we do a redirect to the homepage (lines 13-15) so the request can be submitted again with the proper data. After obtaining the files, we check if the putative image files are images in a supported format (lines 19-22). The extensions allowed are JPG, PNG, and JPEG. We use a variable *extension*, which is initially set to *True*, and its value changes in case the function `check_extension` returns *False* for at least one of the images. In that case, the user is redirected to the homepage to make a new request (lines 24-25).

Assuming every image passes the extension check, we verify the integrity of every image file, save them in the proper directory, and gather their names in a sentence, (lines 26-31). Through the *redirect* procedure (lines 34-35) all of this data is then passed to the function `loading_results` (Listing 5.3).

Since we integrated three models into our web application, each image will be given three classifications, one per model. Moreover, each classification consists of a list of the top 5 results determined by the model. We decided to store all the information returned by the models for the submitted images in a lightweight, local, SQL database implemented using SQLite3. Figure 5.9 illustrates the scheme we used to store the information. The table Tests the data submitted by the user as well as the general information about the tests, such as the *date* and the *ID*. Table Classifications stores the outputs of every model, and the reference to the image tested (red circle).


Tests

ID	Filename	User	Date	Species	IMG	Notes
1	image.png	user1	22 - 04 - 2020	Acacia Longifolia	blob data type	null

Classifications

ID	Test ID	Model	Classification 1	Confidence Level 1	...	Classification 5	Confidence Level 5
1	1	Flo	Acacia prominens	42		Acacia longifolia	2
2	1	iNat	Acacia longifolia	29		-	-
3	1	iNatflo	Acacia longifolia	56		-	-

Figure 5.9: Database tables.

Now that we clarified how and where we store the information, below is displayed the `loading_results` function, mentioned before, in which we execute the tests with the images and store the information in the tables shown in Figure 5.9.

```python
@app.route('/results/<names>/<f_name>/<expert_label>/<comments>',
    methods=['GET','POST'])
def loading_results(names,f_name,expert_label,comments):
    if request.method == 'GET':
        conn = connect_to_db()
        cursor = conn.cursor()
        today = date.today()
        names = names.split(',')
        del names[-1]
        rowspan = len(names)
        inat = []
        fl_inat = []
        flora = []
        index = [1,1,2,2,3,3,4,4,5,5]
        for name in names:
            id = get_id(cursor,'tests')
            expert_id = f_name
```

```
17              filename = app.config["IMAGE_UPLOADS"] + '/' + name
18              thumbnail_name = app.config["IMAGE_THUMBNAILS"] + '/' + name
19              generate_thumbnail(filename,thumbnail_name)
20              img_blob = converttoBinary(filename)
21              data_tests =
                    (id,name,expert_id,str(today),expert_label,img_blob,comments)
22              insert_in_db(conn,'tests',data_tests)
23
24              for i in range(3):
25                  model = Model(MODEL_FILES[i], LABEL_FILES[i])
26                  output = model.classify(MODEL_NAMES[i],filename)
27                  while(len(output) < 12):
28                      output.append("")
29                  if MODEL_NAMES[i] == "iNaturalist":
30                      inat.append(output)
31                  elif MODEL_NAMES[i] == "Flora_On_and_iNat":
32                      fl_inat.append(output)
33                  else:
34                      flora.append(output)
35                  classif_id = get_id(cursor,'classifications')
36                  data_classifications = (classif_id,id,MODEL_NAMES[i],output[2],\
37                      output[3],output[4],output[5],output[6],output[7],\
38                          output[8],output[9],output[10],output[11])
39                  insert_in_db(conn,'classifications',data_classifications)
40          conn.commit()
41          conn.close()
42      if request.method == 'POST':
43          return redirect(url_for('upload_image'))
44      return
            render_template("results.html",index=index,row_span=rowspan,inat=inat,\
45          fl_inat=fl_inat,flora=flora,names=names,\
46              f_name=f_name,expert_label=expert_label,comments=comments)
```

Listing 5.3: Function `loading_results`.

In this first section of the function we do a little pre-processing, (lines 4-13) initiate the connection with the database, retrieve the date, and store the names of the images in a proper way to be used later in the function. Arrays *inat*, *fl_inat*, and *flora* is where we store the results of the respective models, iNat, iNatflo, and Flo. The index array is to be used later when presenting the results.

Now, for each image (line 14) we extract every field needed to fill the Tests table (lines 15-21). First, we get the *ID*, which is unique per image. Afterwards, we generate a thumbnail of the image to be used later for displaying the results. We do this to present the results uniformly. The function takes as arguments the location of the original image and the location where the thumbnail is to be stored. Variables *filename* and *thumbnail_name* hold those same arguments respectively.

To store the image in the database it is necessary to convert it BLOB (line 20). The variable *img_blob* holds that conversion. Now that we have all the information about the image in the proper data types, we use the variable *data_tests* to hold them together and the function `insert_in_db` (Listing 5.4) to insert them to the database.

```python
def insert_in_db(conn,db,query):
    cursor = conn.cursor()
    if db == 'tests':
        try:
            cursor.execute('''INSERT INTO tests
            VALUES (?,?,?,?,?,?,?)''',query)
            conn.commit()
        except sqlite3.Error as error:
            print('Failed to insert data in table tests - ', error)
    elif db == 'classifications':
        try:
            cursor.execute('''INSERT INTO classifications
            VALUES (?,?,?,?,?,?,?,?,?,?,?,?,?)''',query)
            conn.commit()
        except sqlite3.Error as error:
            print('Failed to insert data in table classifications - ', error)
    else:
        print('Error: table {} not found!'.format(db))
```

Listing 5.4: Function to insert data in the database.

We use this function to fill both the *Tests* and *Classifications* tables. The variable *conn* holds the connection to the database, *db* is the table to fill, and *query* is the information to be stored.

Back to the `loading_results` function (Listing 5.3), lines 24-39 is the section where we test the image with each of the three models mentioned, gather the results, and store them in the *classifications* table of the database. We start by creating a variable of the class Model, for that we need to supply two arguments: the TFLite file, which contains the model, and a text file. This last file contains every label that the model was trained to identify. Once done, we can use the function *classify*, of this class, to get the classifications for the images from the model. This function returns the name of the model that identified the picture, the name of the picture, and five classification - confidence pairs provided by the model. This information is stored in one variable, *output*, and later inserted in the database. Sometimes, when models happen to be very confident in the first classification, they return less than five classification - confidence pairs. In that case, we fill the gaps until the *output* variable has the right number of arguments to be stored in the database.

To later present the results to the user, we organize the results of the tests per model. For that, we separate them into three arrays, one per model. Finally, the last step is to obtain the IDs for each of the classifications. We accomplish that by using the function *get_it*, just like we did to obtain the IDs for the tests. To insert in the table classifications of the database, we gather all the information to fill the table in one variable, *data_classifications*, and send it to the function `insert_in_db` (Listing 5.4).

After completing this process for every picture submitted, all that is left is to send the information to the HTML page (lines 44-46). *Index* and *row_span* are just details to help put the information in the right place. The arrays *inat*, *fl_inat*, and *flora* contain the results of the tests performed on the images according to the models iNat, iNatflo, and Flo respectively. The array *names* holds the names of the images submitted. *f_name*, *expert_label*, and *comments* contain the data with which users filled the homepage form (Figure 5.5). Finally, Listing 5.5 shows how we render the data in the *results.html* page. For each picture, we display the filename, a thumbnail of the image, and a table with the tests' results. The top row of the table contains the #. It represents the order of the classifications that each model assigned to the image. Also, in the top row are the names of the three models. The remaining rows display the labels and the respective value of confidence assigned by the models, ordered descending. Once the table is complete we use a JavaScript function to color the classifications that match the *Species* submitted by the user.

```
{%for i in range(row_span)%}
        <h3>{{names[i]}}</h3>
        <img src="{{url_for('static',filename='thumbnails/'+names[i])}}"
            alt="Plant image!"/>
            <table class="customers" id="{{names[i]}}">
                <tr>
                    <th></th>
                    <th>iNaturalist</th>
                    <th>FloraOn</th>
                    <th>iNaturalist + FloraOn</th>
                </tr>
                {% for j in range(0,9,2)%}
                    <tr>
                        <td>{{index[j]}}</td>
                        <td>{{inat[i][j+2]}} - {{inat[i][j+3]}}</td>
                        <td>{{flora[i][j+2]}} - {{flora[i][j+3]}}</td>
                        <td>{{fl_inat[i][j+2]}} - {{fl_inat[i][j+3]}}</td>
                    </tr>
                {% endfor %}
                    <script type="text/javascript">
            var id = "{{names[i]}}";
            var label = "{{expert_label}}";
            console.log(id,label);
            get_rows(id,label);
          </script>
            </table>
        <hr>
        {% endfor %}
```

Listing 5.5: Data rendering.

Chapter 6

Evaluation

In this chapter we provide an evaluation of the three flora identification models we developed, using test images that form three test groups, including one composed by images provided by a domain specialist. The specialist test group is also used for a comparison with the PlantNet identification service. We first describe the evaluation methodology (6.1) in terms of models, test groups, and evaluation metrics. We then present comparative results for the three models (6.2). The chapter ends with the comparative assessment to PlantNet (6.3).

6.1 Methodology

6.1.1 Models

Table 6.1: Model characteristics.

Model	Dataset	Images	Species	Genus
Flo	FloraOn	16633	1124	574
iNat	iNaturalist PT & Spain	37478	625	192
iNatflo	iNaturalist PT + FloraOn	36758	1427	659

As explained in the previous chapter, we derived three different models trained with distinct image sets: FloraOn photos, iNaturalist photos, and a combination of FloraOn and iNaturalist photos, identified in this chapter as **Flo**, **iNat** and **iNatflo**, respectively. The characteristics of the models are summarized in Table 6.1 in terms of the number of images, species and genus.

Comparing the Flo and iNat model, we can observe that the Flo model is built using much fewer images (45% of the images used in iNat; only 15 images per species on average vs. 58 in iNat), but covers a much greater number of species (by a factor of approx. 2) and genus (by a factor of approx. 3). The image set of Flo is also more balanced in terms of distribution of photos per species, as we highlight later in this section. By combining iNaturalist and FloraOn images, the iNatFlo model tries to cover as many species as possible and overcome some of limitations in data volume of the Flo model.

6.1.2 Test groups

Table 6.2: Test group characteristics.

Test group	Images	Species	Genus
FloraOn	919	316	229
iNaturalist	4293	316	229
Specialist	219	28	25

To test the models generated we prepared 3 image data sets, called test groups. The first two test groups, identified as **FloraOn** and **iNaturalist**, consist of a subset the images that Google Auto ML reserved in the validation and test phases of the Flo and iNat models. These images are identified in a CSV file that can obtained from Google Auto ML, and the subset we took contained only species that occurred in all the models for a fair comparison. The third test group, identified as **Specialist**, consists of a set images supplied by a domain specialist from Parque Biológico de Gaia. The Specialist test group allows us to have a set of images that was independent of FloraOn or iNaturalist. More subjectively, this provides us with a test group that may be less biased in "style", as FloraOn photos tend to have high-quality and be close-up, iNaturalist photos have varying quality, and the Specialist photos tend to be close-up but have lower quality than FloraOn.

The characteristics of all test groups are summarised in Table 6.2. The difference in numbers for the FloraOn and iNaturalist test groups is in direct relation with the volume and variety we highlighted earlier for the iNat and Flo models. The Specialist test group is much smaller than the other two in terms of images, species or genus.

6.1.3 Evaluation metrics

We conducted the base evaluation of our models by submitting images from the test groups to each model, and calculating the following metrics:

- **Top-1** and **Top-5**: the percentage of images for which a model yields the correct species in first place or in the first 5 places, respectively, providing measures of classification accuracy;
- **Average confidence level**: the average value of the confidence level that are reported by the model for images in Top-1 and Top-5, providing a measure of classification confidence by the model;
- **Precision** and **recall**: a model's precision and recall, expressed as usual by the ratio between the number of true positives (TP) and, in the case of precision, the sum of true

and false positives (FP), or, in the case of recall, the ratio between the number of true positives (TP) and the sum of true and false negatives (FN)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

By definition, a high precision indicates a low fraction of False Positives (FPs) results, and a high recall indicates a low fraction of False Negatives (FNs) results.

6.2 Model results

6.2.1 Top-1 and Top-5

Figure 6.1 depicts the Top-1 (6.1a) and Top-5 (6.1b) of all models across for every test group. The iNatflo model has noticeably the best performance both in terms of Top-1 and Top-5, except for the Specialist test group for which the iNat model performance is approximately the same. For the iNatflo model, the results are very good for the the Flo and iNaturalist test group. For other models, as expected, Flo is better than iNat for the FloraOn test group, and iNat is better than Flo for the iNaturalist test group. The Top-1 and Top-5 numbers are only higher than 50 % for the FloraOn test group.

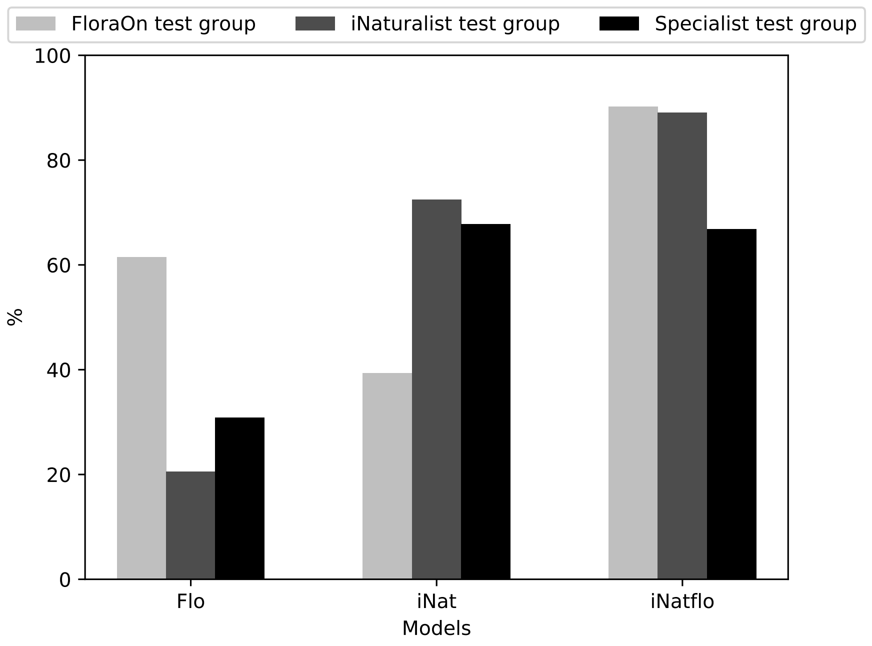
6.2.2 Confidence levels

We also measured the average confidence level, as depicted in Figure 6.2, for the Top-1 (6.2a) and Top-5 (6.2b) results returned by each model for each test group. The results are in line with the base Top-1/Top-5 results, with the best results observed overall for iNatflo and the worst for FloraOn. It is also noticeable that the Flo model has low confidence levels ($< 50\%$) even for the FloraOn test group. We believe this is explained by the fact that the Flo model is trained with relatively few images per species, while at the same time covering a wide number of species.

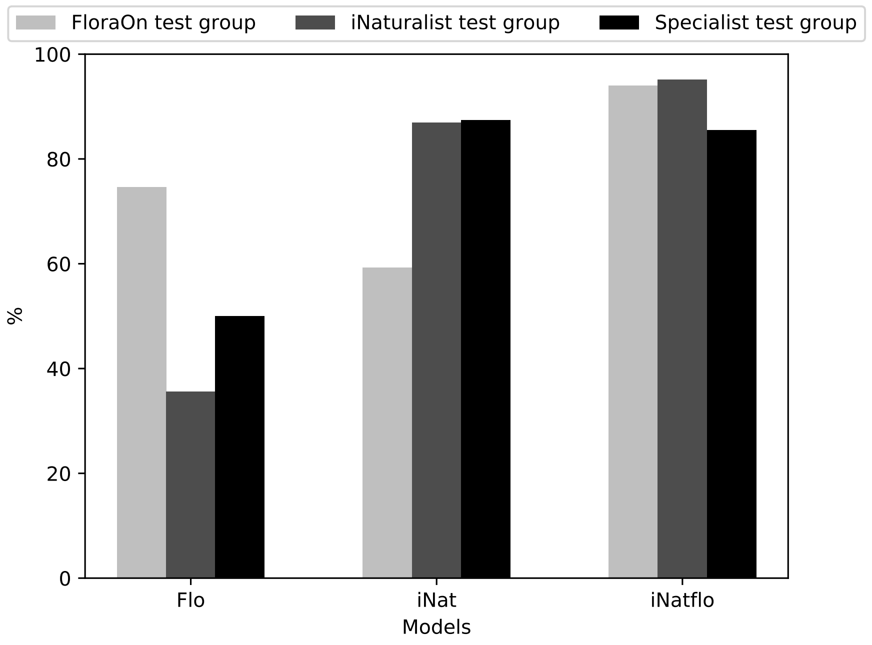
6.2.3 Top-1 and sensitivity to train set

The correlation between the number of training images and Top-1 performance is put in perspective in the scatter plots of Figure 6.3, where for each test group (FloraOn,6.3a; iNaturalist 6.3b; Specialist, 6.3c). A point is depicted for each species and model (Flo in white, iNat in gray, and iNatflo in black) in each of the test groups. The different sizes of the markers reflect the number of tests for a species in the test group.

The expected the trend is that a model's performance tends to correlate with the number of training images, and the plots confirm it. Mainly in Figures 6.3b and 6.3c, we can see that



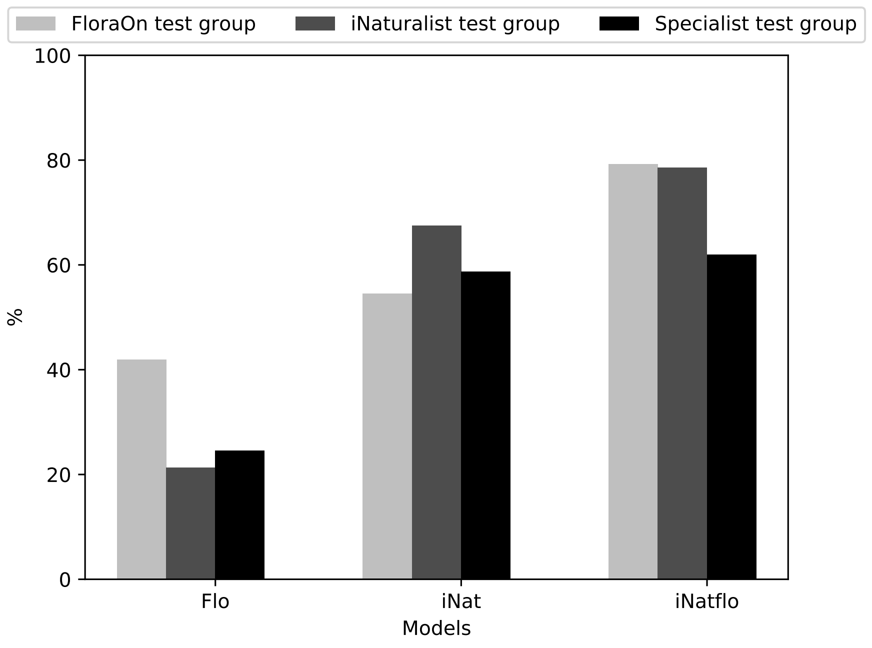
(a) Top-1



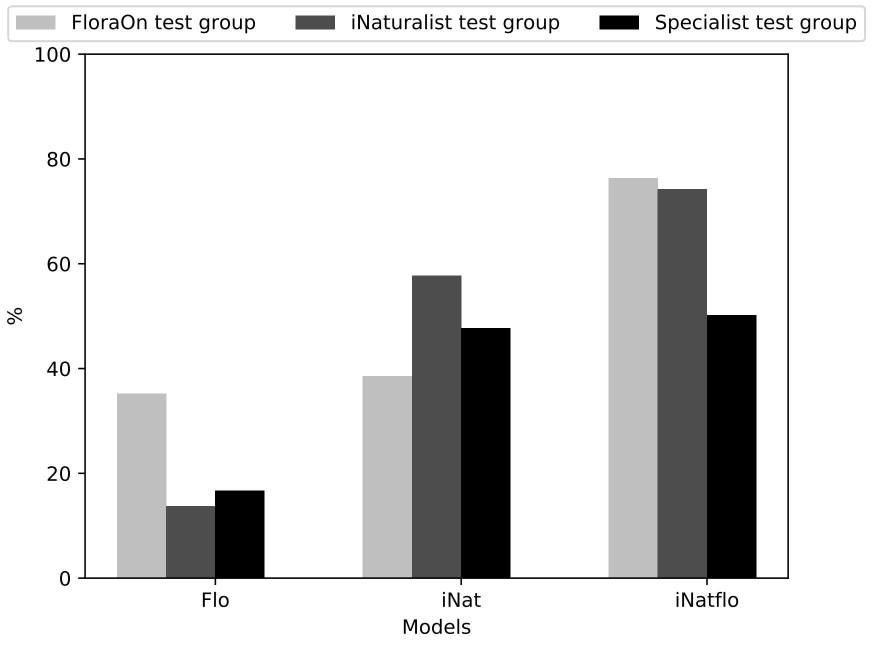
(b) Top-5

Figure 6.1: Results for the Top-1 and Top-5 metrics.

the models identify more easily species that have more training images, though not always. In particular, the number of test images can be low for some species, especially in the Flo and Specialist test groups, hence the Top-1 performance can degrade easily in these cases. For instance, in the FloraOn test group, many species have only one or two tests, so, a model may get both tests correct, only one or neither, which explains the three lines in the plot for the



(a) Top 1



(b) Top 5

Figure 6.2: Average confidence level for Top-1 and Top-5 results.

values 0, 50, and 100. In the iNaturalist test group, Figure 6.3b, the scenario is different from the previous, since the number of tests for most species is considerably higher it gives a better perspective of how each model performs. The Flo model produces results considerably worse than the others. Both iNat and iNatflo models successfully identified most species in this test group. Highlighted in this plot is the difference between the number of training images of the models. As we can see, all species from the Flo model appear in the left part of the plot, while most species from the iNat and iNatflo models appear in the middle and right parts. The data from the Specialist test group, illustrated in Figure 6.3c, is similar to the iNaturalist test group. The Flo model performs poorly while iNat and iNatflo maintain their positive trend.

The fact that Flo possesses the widest scope of species and only a few training images, for each, makes it more vulnerable to mistakes when classifying images. Besides these, other aspects that can cause the model to be misleading is the kind of image submitted, photo style, and photo quality (e.g., focus, light, noise). We considered three kinds of photo style: **(M)** macro/closeup of plant detail; **(N)** normal view with full plant and not much else; and **(W)** wide angle view with plant included in habitat. Even species with a considerable number of training images can cause problems for the models when identifying them, depending on the mix of M, N, and W style photos. For example, in the iNaturalist test group (Figure 6.3b) there can be spotted five species that, although have a reasonable number of training images, the model iNat had difficulty when identifying them. Table 6.3 provides the plot numbers of the mentioned species.

Table 6.3: iNat species with unexpected low test results in the iNaturalist test group.

Species	Training Images	Tests	Top 1 (%)
Quercus faginea	53	14	14
Sonchus oleraceus	59	15	27
Capsella bursa-pastoris	65	16	38
Rubia peregrina	80	20	30
Myrtus communis	78	20	40

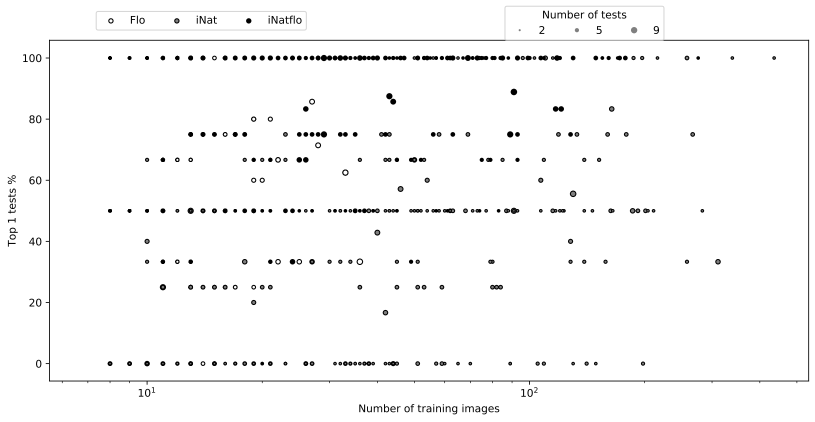
To better understand these results we analyzed the kind of images, i.e., photo style, used for testing as well as in training. Table 6.4 describes the photo style of the images used when testing these species, while Table 6.5 shows the photo style of the images used in training.

Table 6.4: Test images - photo style.

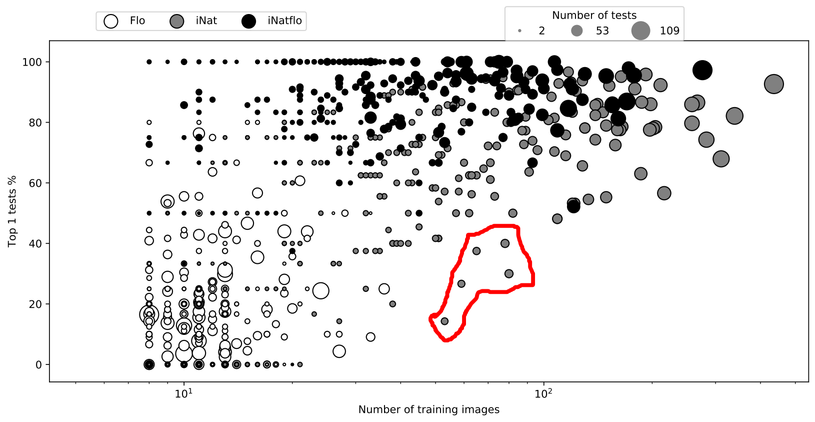
Species	Tests	M	N	W
Quercus faginea	14	9	2	3
Sonchus oleraceus	15	8	4	3
Capsella bursa-pastoris	16	7	5	4
Rubia peregrina	20	15	5	0
Myrtus communis	20	12	7	1

Table 6.5: Training images - photo style.

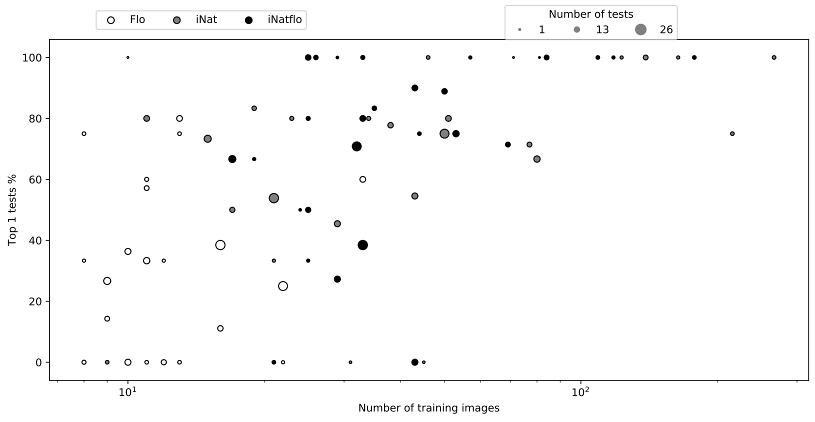
Species	Training Images	M	N	W
Quercus faginea	53	38	7	8
Sonchus oleraceus	59	39	18	2
Capsella bursa-pastoris	65	39	13	13
Rubia peregrina	80	35	30	15
Myrtus communis	78	52	24	2



(a) FloraOn test group



(b) iNaturalist test group



(c) Specialist test group

Figure 6.3: Correlation between Top-1 score and the number of training images per species across the different test groups.

Considering the number of images used in training (Table 6.3) for each of these species we expected the model iNat to have better results than the ones presented. Our best guess is that one of the problems is the images with wide angle view of the plants (W), both in training and

Table 6.6: Tested images - correctly identified according to photo style.

Species	M	N	W
Quercus faginea	3/9	0/2	0/3
Sonchus oleraceus	4/8	0/4	0/3
Capsella bursa-pastoris	4/7	2/5	1/4
Rubia peregrina	5/15	1/5	0/0
Myrtus communis	6/12	2/7	0/1

testing. These images have very little to no detail, and the environment surrounding the plants differs quite a bit. Although we firmly believe this to be true, it is just a hypothesis, and due to limited time, we could not do a more in-depth investigation to prove it.

For the following species, we describe what caused the model trouble when identifying them.

With regards to *Quercus faginea*, there are lots of species within the *Quercus* genus. Not only is the physiognomy very similar, but also, the species of this genus tend to interbreed freely, giving way to specimens with a mix of features hard to identify even for specialists. So even with close-up images (M), the model tends to be confused between different *Quercus* species.

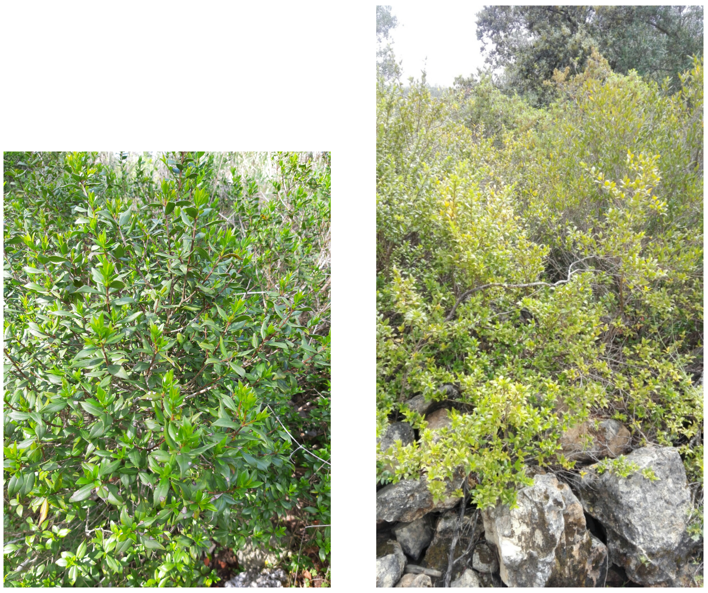
The flower of *Sonchus oleraceus* is similar to a variety of species including *Sonchus tenerrimus* and *Sonchus asper*. In images less detailed, the flower of this species can also be mistaken with the flowers of several species of the family *Asteraceae*, a difficult subject even for specialists. These similarities with other species caused some doubts in our model, leading to not very good results.

Finally, *Myrtus communis* has very distinctive features, that being the flower and the leaf. The model performed well when identifying good quality and detailed images. The low-quality images and the surroundings of the plant were the main factors that led to not so good results. Figures 6.4a and 6.4b are examples of some bad images used for testing this species.

6.2.4 Precision and Recall

The precision and recall values for our three models are shown in Table 6.7 for confidence level thresholds of 10, 25, and 50 %. Per each confidence threshold, and for both precision and recall, the best performance is identified in bold, and the worst in italic. To calculate precision and recall per model, test group, and threshold, we considered true positives to be the correct (first) classifications returned by the model with a confidence higher than the threshold. If the confidence is higher than the threshold but the classification is incorrect, we considered an image to be a false positive for the species returned as a result. For confidence levels lower than the threshold, an image is considered a false negative in any case.

Looking at the overall numbers, Flo has good precision values but very average, and sometimes poor values when it comes to recall. It means that most of the tests that it identifies as correct are correct, but it does not recognize a large portion of tests that it should. iNat has excellent



(a) Low quality image. (b) Plant surroundings.

Figure 6.4: Images with bad attributes.

precision values, most positives being true, while recall falls a bit short, identifying just around fifty percent of tests that it should. iNatflo has, once again, better numbers overall.

Precision and recall follow the expected behavior: as larger values for threshold are considered, precision worsens (due to increasing FPs) whereas recall improves (due to decreasing of FNs).

6.3 Comparison with PlantNet

We performed a comparison of our work with the PlantNet mdoel using the publicly available RESTful Web service [30], by submitting photos through this service and comparing the obtained results with the ones we obtained. For the same purpose, we also considered the use of the iNaturalist API [16], but the image classification functionality that happens to be used internally by the iNaturalist mobile app but is not available for public use.

The PlantNet service usage is limited to 50 species classification a day, hence, for a timely evaluation, we decided to perform a comparison only for the Specialist test group (219 photos of 28 species). We also perform a comparison with our iNatflo model only, given that it has the best overall results and is the one that covers more species. Even if this is a limited comparison, it provides us with an impression of how our work compares with a state-of-the-art mature system.

The results are shown in Table 6.8, in terms of the Top-1 and Top-5 values for both the iNatflo model and the PlantNet classification. The global values for the entire test group are listed in the first row, and subsequent rows provide individual results by species. We display species names in bold in the cases where iNatflo outperforms Plantnet, in italic when the opposite happens, and in the regular font when the performance is similar. From the results, we can

Table 6.7: Results for precision and recall.

Test group	Model	Conf. Level (%)	Precision (%)	Recall (%)
FloraOn	Flo	50	**98**	38
		25	92	49
		10	83	56
	iNat	50	*78*	*28*
		25	*63*	*35*
		10	*46*	*40*
	iNatflo	50	**98**	**89**
		25	**96**	**90**
		10	**93**	**91**
iNaturalist	Flo	50	*81*	*3*
		25	*65*	*8*
		10	*42*	*16*
	iNat	50	94	68
		25	85	79
		10	76	92
	iNatflo	50	**98**	**89**
		25	**95**	**92**
		10	**91**	**97**
Specialist	Flo	50	*84*	*7*
		25	*73*	*16*
		10	*59*	*25*
	iNat	50	**90**	57
		25	80	**76**
		10	71	91
	iNatflo	50	85	**63**
		25	**82**	73
		10	**72**	**98**

observe that the iNatflo performs worst overall than PlantNet, with a lower Top-1 value, 67 % vs. 78 %, but only slightly lower Top-5 value, 87 % vs. 88 %. Regarding individual species, iNatflo performs better than Plantnet in 6 cases, worse in 13 cases, and there is a tie in 10 cases. Even if the results of our model are worse, we find them encouraging, considering that our model is only a research prototype while PlantNet is an established system.

Table 6.8: iNatFlo vs. PlantNet – Top-1 and Top-5

	Top-1 (%)		Top-5 (%)		
Species	iNatflo	PlantNet	iNatflo	PlantNet	Tests
All	67	78	87	88	219
Anthyllis vulneraria	71	83	79	96	24
Beta maritima	50	0	50	0	8
Borago officinalis	100	100	100	100	7
Cakile maritima	100	0	100	100	3
Centaurea sphaerocephala	90	90	100	100	10
Cistus salviifolius	100	100	100	100	4
Crataegus monogyna	100	100	100	100	4
Crucianella maritima	0	100	64	100	11
Cyperus capitatus	80	100	100	100	5
Davallia canarienses	0	0	0	30	10
Digitalis purpurea	100	100	100	100	3
Dryopteris affinis	67	33	100	33	3
Erica australis	100	100	100	100	1
Erica umbellata	75	100	100	100	4
Euphorbia paralias	75	83	83	100	12
Glaucium flavum	100	100	100	100	9
Malcolmia littorea	71	86	86	100	7
Medicago marina	100	100	100	100	5
Narcissus triandrus	80	70	90	100	10
Polygonatum odoratum	67	87	100	100	15
Polystichum setiferum	50	0	50	50	2
Saxifraga spathularis	0	0	75	50	4
Scilla monophyllos	89	89	89	100	9
Serapias cordigera	100	100	100	100	6
Seseli tortuosum	83	83	100	100	6
Silene littorea	38	53	69	67	15
Silene uniflora	27	100	73	100	11
Tulipa sylvestris	63	88	75	88	8
Ulex minor	33	67	67	100	3

Chapter 7

Conclusions

In this dissertation, we present a web application that uses deep learning techniques to identify plants in photographs submitted by users, focusing on the Portuguese flora. Using data from iNaturalist and Flora-On we developed three models using Google AutoML. The models were then integrated into the Web application. First, we compared their performance with data from three different test groups. Afterwards, the one with better performance was tested against a state-of-the-art model from PlantNet. iNaturalist data is not very controlled and it is heterogeneous, and, specifically, the photographs vary widely in quantity and quality. The resulting model misses the identification unacceptably. Flora-On has more reliable data and excellent quality photographs but few examples per species, resulting in a model that performs poorly when compared to iNaturalist. The third model, with data from both sources, proved to be the most efficient.

The Web application was built so that we could have a better evaluation of the models developed. Part of the future work is to improve the interface. We also intend to develop models with different CNN architectures to test response time and results, when classifying images. We used the photographs as taken from iNaturalist and Flora-On, without any further treatment. Experience tells us that selecting adequate parts of a photograph, e.g., isolating as much as possible the target from the background can have a significant impact on the accuracy of the final model. A revision of the datasets can, therefore, we believe, improve our current results.

Bibliography

[1] Flora de Portugal Interactiva. (2014). Sociedade Portuguesa de Botânica. www.flora-on.pt. Accessed September 2020.

[2] A. Affouard, H. Goëau, P. Bonnet, J.-C. Lombardo, and A. Joly. Plantnet app in the era of deep learning. In *International Conference on Learning Representation*, 2017.

[3] The Cornell Lab of Ornithology. https://www.birds.cornell.edu/home. Accessed September 2020.

[4] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *IEEE conference on Computer Vision and Pattern Recognition*, pages 2147–2154, 2014.

[5] D. Fagella. The rise of neural networks and deep learning in our everyday lives – a conversation with yoshua bengio. https://emerj.com/ai-podcast-interviews/the-rise-of-neural-networks-and-deep-learning-in-our-everyday-lives-a-conversation-with-yoshua-bengio/, 2016.

[6] Flask. https://flask.palletsprojects.com/en/1.1.x/. Accessed August 2020.

[7] K. J. Gaston and J. I. Spicer. *Biodiversity: an introduction*. John Wiley & Sons, 2013.

[8] Global Biodiversity Information Facility. https://www.gbif.org/. Accessed September 2020.

[9] Google Cloud Storage. https://cloud.google.com/storage. Accessed July 2020.

[10] Google Auto ML – Cloud Vision. https://cloud.google.com/vision/overview/docs#automl-vision. Accessed July 2020.

[11] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *International conference on Machine Learning*, pages 1764–1772, 2014.

[12] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[13] K. ichi Ueda. inaturalist research-grade observations. GBIF.org - https://doi.org/10.15468/ab3s5x, 2020.

[14] Imagenet. http://www.image-net.org/. Accessed September 2020.

[15] iNaturalist. https://www.inaturalist.org/. Accessed September 2020.

[16] iNaturalist API. https://api.inaturalist.org/. Accessed July 2020.

[17] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis. Machine learning applications in cancer prognosis and prediction. *Computational and Structural Biotechnology Journal*, 13:8–17, 2015.

[18] W. F. Laurance. Habitat destruction: death by a thousand cuts. *Conservation biology for all*, 1(9):73–88, 2010.

[19] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[20] Leps. https://leps.it/. Accessed September 2020.

[21] M. W. Libbrecht and W. S. Noble. Machine learning applications in genetics and genomics. *Nature Reviews Genetics*, 16(6):321–332, 2015.

[22] C. Linnaeus. *Systema naturae*. Stockholm Laurentii Salvii, 1758.

[23] E. Mayr. *The growth of biological thought: Diversity, evolution, and inheritance*. Harvard University Press, 1982.

[24] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. A machine learning approach to building domain-specific search engines. In *IJCAI*, volume 99, pages 662–667. Citeseer, 1999.

[25] B. McKelvey. *Organizational systematics–taxonomy, evolution, classification*. University of California Press, 1982.

[26] D. Michie, D. J. Spiegelhalter, C. Taylor, et al. Machine learning. *Neural and Statistical Classification*, 13, 1994.

[27] Parque Biológico de Gaia. https://www.parquebiologico.pt/. Accessed September 2020.

[28] S. L. Pimm and P. Raven. Biodiversity: extinction by numbers. *Nature*, 403(6772):843, 2000.

[29] PlantNet. https://plantnet.org/. Accessed September 2020.

[30] PlantNet Plant Identification API. https://my.plantnet.org/. Accessed September 2020.

[31] PlantSnap. https://www.plantsnap.com/. Accessed September 2020.

[32] C. D. Preston. Methodus plantarum nova: John ray (1682), 2016.

[33] W. Ride et al. *International code of zoological nomenclature*. International Trust for Zoological Nomenclature, 1999.

[34] L. Roeder. Netron - Visualizer for neural network, deep learning, and machine learning models. https://github.com/lutzroeder/netron. Accessed September 2020.

[35] Y. Roskov, T. Kunze, L. Paglinawan, T. Orrell, D. Nicolson, A. Culham, N. Bailly, P. Kirk, T. Bourgoin, G. Baillargeon, et al. Species 2000 & itis catalogue of life, 2013 annual checklist. *Species 2000/ITIS*, 2013.

[36] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443. Springer, 2006.

[37] R. Santos. Plataforma de Monitorização da Biodiversidade no Concelho de Gaia, 2019. Faculdade de Ciências da Universidade do Porto.

[38] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[39] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern recognition*, pages 1–9, 2015.

[40] Tensorflow lite. https://www.tensorflow.org/lite/. Accessed September 2020.

[41] Tensorflow lite – image classification demo. https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android. Accessed September 2020.

[42] J. d. Tournefort. *Institutiones rei herbariae.* 1700.

[43] G. Van Horn, O. Mac Aodha, Y. Song, Y. Cui, C. Sun, A. Shepard, H. Adam, P. Perona, and S. Belongie. The iNaturalist species classification and detection dataset. In *IEEE conference on Computer Vision and Pattern Recognition*, pages 8769–8778, 2018.

[44] T. Watson. percent of earth's species still unknown. *National Geographic News*, 86.

[45] Wikipedia. Convolutional Neural Network. en.wikipedia.org/wiki/Convolutional_neural_network. Accessed September 2020.

[46] C. Zhang and Y. Ma. *Ensemble machine learning: methods and applications.* Springer, 2012.