Tree Species Identification Using Convolutional Neural Networks for Mobile Applications

Practical Machine Learning

Diogo Gomes nº 26843

Janice Rodrigues nº 25902

Rafael Oliveira nº 26066

U LISBOA | UNIVERSIDADE DE LISBOA

INSTITUTO SUPERIOR D AGRONOMIA

Introduction

Biodiversity Databases are important for scientific progress as they constitute a central repository of species data, helping scientists track species distributions, monitor environmental changes, support conservation efforts, and guide policy decisions, in a way that is quick and globally accessible. The biodiversity databases are in an ever-lasting quest to acquire more information and be more complete, however this progress creates a problem: There aren't many experts that can, without a doubt, correctly identify a specific specimen that is presented to them, in a scale that is compatible with the platforms needs. Some platforms like the *GBIF*, *PlantNET* and *iNaturalist* tackle this challenge by operating classification models with supervision from user-submitted feedback to correctly label the observations allowing more users to create occurrences. After analysing these platforms and some related apps we've reached the conclusion that this market has a fundamental flaw, that is inhibiting their growth: There is little incentive for the casual users to contribute or invest time into research for these endeavours. We noted that it mostly only relies on information on the species, discussion forums and info on plant health status (which in a nature activities context isn't very useful), in short, the user-experience is very dull, uninspiring and most importantly uninteresting.

To capture this market, in this project, we intent to present and set up the groundworks for the development on a mobile application that allows the users to collect vegetation species information and to send those occurrences to biodiversity databases in a way that promotes and rewards the user's actions. The backbone of this app will be a machine learning model that allows for the species classification of user-taken images in a way that is accessible to everyone. This app will also have features such as nature trails, friend lists, cosmetics, an algorithm driven global message board, to incentivise the user's activity on the app (see App_Mockup.md file). In this report we will not focus on the app development, only on the creation of the aforementioned Image Classification Model.

We will develop the image classification model only on the 3 most common species in Portuguese forests (Eucalyptus globulus, Pinus pinaster and Quercus suber) as a proof of concept that can be later expanded to more species, including herbaceous species and plants found only outside the country.

We will compare the performance of two models, one self-built Convolution Neural Network ("CNN model") and another based on ResNet18 foundation model ("ResNet model").

A Convolutional Neural Network (CNN) is a type of machine learning model that takes direct inspiration from the way the human brain analyses images, allowing them to autonomously detect patterns and features in data. This makes them especially powerful in tasks like image classification, object detection, and facial recognition. At the core of a CNN are convolutional layers, which use small filters that move over the input image to detect features like edges, textures, and shapes. Early layers might detect simple elements like lines

or corners, while deeper layers combine these into more complex structures like eyes, faces, objects, etc. They also greatly reduce the number of parameters compared to traditional Neural networks, making them more efficient for high-dimensional inputs like images.

The ResNet foundation model is a publicly available deep CNN model that was pretrained on a public dataset called ImageNet that contains around 1.2 million images and 1000 output classes like cats, houses, planes, trees, etc. It was developed by Microsoft in 2015, and it introduced several breakthroughs in deep learning. There are five versions of ResNet available depending on the number of layers they contain, ranging from ResNet-18, with a total of 11 million parameters, to ResNet-152, with 60 million parameters. The higher the number of layers, the more accurate the model is, but also the more computer power it uses. These models are typically used for image classification and can be fine-tuned to allow their use in more specific purposes and greater number of output classes.

Data

To create the tree image classification model, the data consisted of labelled RGB images of the previously mentioned species, that have been scrapped from GBIF using a python script. We had to make sure that the extracted images were either JPG or JPEG files, sourced from Portugal and that they all fall under a public domain license (cc0 1.0) or under the cc by 4.0 license, so that we could freely use the data for our intents and purposes.

After manually filtering the photos for high-definition images which clearly display the texture and shape of the tree and its leaves, there were 593 images with varying dimensions labelled as "*species_0000*", totalizing 145 from Eucalyptus globulus, 216 from Pinus pinaster and 229 from Quercus suber trees.

For both models, all images we resized into a 128x128 pixel shape and then were transformed into tensors to be correctly read by the model. The images used for training we also randomly flipped them, rotated them by 15 degrees at the maximum and changed their brightness and contrast by 20% at the maximum. This was conducted to increase the variability of the dataset, forcing the model to learn more general and meaningful patterns, preventing the overfitting of the data, leading to better performance on unseen natural data. Since the test data should represent real data and be used to assess the model's performance with real data, this data augmentation is not needed.

Data Organisation

To test the model's performance at each epoch we used Simple Division Validation, this mean that in both models, 80% of the data was separated into train data to fit the models, and 20% to then calculate their accuracy at each epoch. We applied a parameter to guaranty

that the proportions of each label in the divided sets are the same as the original ratio. These training images were loaded into the model in batches of 16, promoting efficiency, optimizing performance, and increasing model stability.

Method

We started by developing the Convolution Neural Network, machine learning model from scratch. This model is made up by a feature extraction phase and then a classification phase. The feature extraction happens in three chunks, the first takes in a 128x128 RGB image and then there are two sets of a 32-filter convolution followed by a normalization and a ReLu non-linear function. At the end of each chunk the output images get quarter-size smaller until it is only 32x32 pixels and has 128 convolution filters. At the end of each chunk some neurons also get turned off randomly to prevent overfitting. To classify, this 32x32x128 element gets flattened into a 1-dimensional vector with 131,072 values that is flattened into only 256 values. It then applies a ReLu function and turns off some neurons again, so that it can finally transform those values into 3, one for each of our labels. Each one of these values corresponds to the probabilities that the model assigns to each label, considering a given image input.

```python
class DeeperCNN(nn.Module):
    def __init__(self, num_classes):
        super(DeeperCNN, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.Conv2d(32, 32, kernel_size=3, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),  # 128 → 64
            nn.Dropout(0.25),

            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(2),  # 64 → 32
            nn.Dropout(0.25),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Dropout(0.25),
        )

        self.classifier = nn.Sequential(
            nn.Flatten(),
            nn.Linear(128 * 32 * 32, 256),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(256, num_classes)
        )

    def forward(self, x):
        x = self.features(x)
        x = self.classifier(x)
        return x
```

Figure 1 – Architecture of the CNN Model

As for the ResNet model, it begins with an initial convolutional layer, followed by a max pooling layer. This initial stage leads into four subsequent stages, each consisting of two "Basic Blocks" the fundamental residual unit of ResNet-18. After the four stages of residual blocks, a global average pooling layer reduces each feature map to a 1x1x512 vector, significantly reducing parameters. At the end, a fully connected layer maps the desired number of output classes, typically followed by a SoftMax activation for classification.

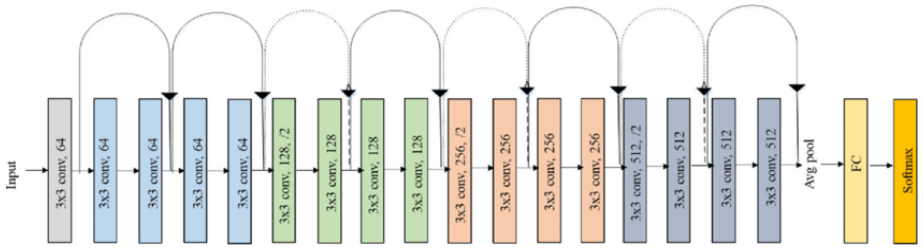

Figure 2 - Diagram of the ResNet18 structure.

For both models, Cross Entropy Loss criterion was used as the loss function, which calculates the difference between the predicted class probabilities and the true class labels, a label weights tensor was also created to apply in the criterion function to convey that the samples for each species were not the same size. And as the optimizer, the ADAM optimizer function was used, which does the gradient descent in order to minimize the loss. The learning rate was set to a small value of 0.00001.

Results and Analysis

First, we will analyse the accuracy results of the CNN model and then the ResNet model, allowing us to compare their performances.

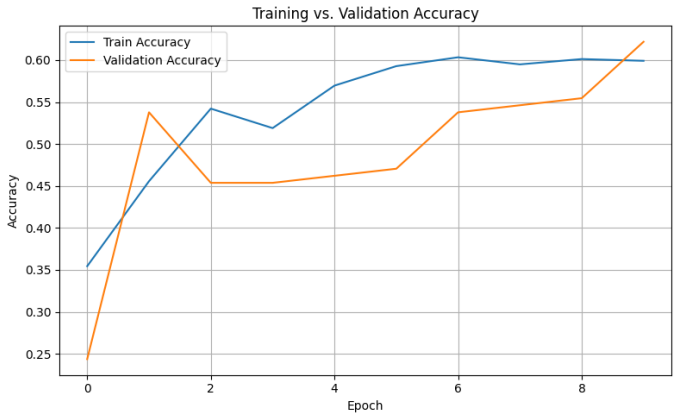

Figure 3 – Accuracy Curves of training and validation sets using the CNN Model

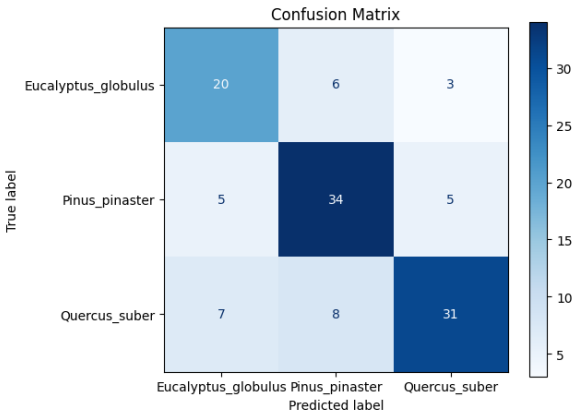

Figure 4 – Confusion Matrix of the validation sets using the CNN Model

As can be seen in Figure 3, at the 10th epoch, our CNN model that was trained and tested just using 600 images has an accuracy of about 60% for both the training and validation sets. That number to most may appear quite small but taking into consideration the sheer amount of visual diversity among plants of the same, we consider this value acceptable, there were close up images, photos of the canopy, from the ground up, photos of a branch on the ground, images of juvenile trees, as an example. We attempted this model multiple times, altering different parameters such as different output sizes as to lose less details, more layers and a reduced learning rate, to improve the accuracy but we couldn't reach values above 65% accuracy.

Looking at the confusion matrix in figure 4, our model produces not-so-satisfactory results when it comes to predicting the right labels, mainly in Pinus pinaster images. Throughout the process of its creation, the model constantly had trouble distinguishing Eucalyptus from Quercus, due to similarities in their tree canopy shape, we suppose, but it could distinguish well when there was a Quercus suber trunk visible. To tackle this problem, we made sure there were more detailed images of the leaves, which were more visually

differentiated, in our dataset but it didn't greatly impact the model's accuracy. To solve this, maybe the available Eucalyptus data should be greatly increased.

We weren't totally contempt with this performance, so we tried a different approach using a pre-trained foundational model, the ResNet18.

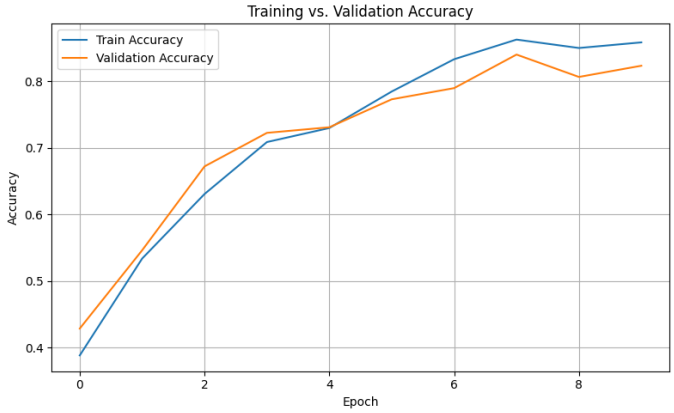

Figure 5 – Accuracy Curves of training and validation sets using the ResNet Model

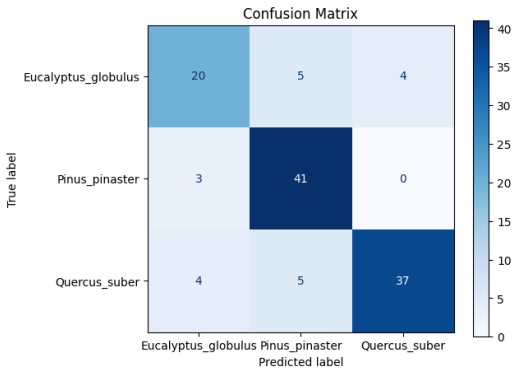

Figure 6 – Confusion Matrix of the validation sets using the ResNet Model

ResNet, as a foundation model, has been trained on a far larger dataset for all of our species, allowing the model to know really well on how to distinguish textures and shapes. This increased the general ability of the model to correctly classify our species as illustrated in the accuracy curves in figure 5, in which it achieved accuracy rates above 80%, much larger than our CNN other model.

Visually, the confusion matrix for both models appears quite similar but looking and the values it is clear that the model using ResNet is much better at identifying the trees, except Eucalyptus which has only a marginal improvement; further illustrating how difficult it is to describe this, very diverse, label.

Next, in table 1, we see the classification report values for both models.

Table 1 – Values extracted from the classification reports of both models

	CNN Model		ResNet Model	
	Precision	**F1- score**	**Precision**	**F1- score**
Eucalyptus globulus	0.62	0.66	0.74	0.71
Pinus pinaster	0.71	0.74	0.80	0.86
Quercus suber	0.71	0.73	0.90	0.85

Both precision and f1-score values in the ResNet model are higher, indicating that it is more accurate than the CNN model as we had stated previously. Most of the F1 values obtained are considered acceptable except that of the eucalyptus in the CNN model which is below 0.7 indicating a need for better data or more fine-tuning.

Deployment

For deployment, the models were saved using torch, generating two files called *model.pth* and *ResNet_model.pth* representing our CNN and the ResNet models, respectively. These files stored the model's weights, which can then be loaded on another computer after establishing the same model structure beforehand.

Then, a *Huggingface* space was created to deploy the models using the *Gradio* library to create the GUI of the space. The space allows the user to change between both models through a list menu and then click on the "Load Model" button. There are nine example images provided for experimentation, but the user can also upload their own photos.

To display the outputs, two boxes are provided. A "Prediction Probabilities" box, where the model provides the likeliness of the input photo to match to each label as a percentage, and a "Final Prediction" box where the model considers the prediction probabilities and only outputs a final answer when there is great certainty of a single label. So, for example, if the probabilities were 38%, 33% and 29%, the "Final Prediction" box will display "Inconclusive, low confidence".

Conclusion

To conclude, even in this small scale, the ResNet model is still somewhat short of our needs, as it failed to correctly label the image in 1 out of every 5. Our CNN model performed even worse and showed great liabilities in almost every, of the few labels. For the final implementation of these models, more images to train them are needed, and perhaps also making them deeper and more robust. One option to achieve this would be to use a deeper ResNet. If implemented, in theory, every new photo taken could be used to further train the model after being correctly labelled by an expert, increasing the dataset size, and inching closer to a "perfect" algorithm.

References

He, K., Zhang, X., Ren, S., & Sun, J. (2015, December 10). Deep residual learning for image recognition. arXiv.org. https://arxiv.org/abs/1512.03385

Deepchecks. (2023, March 22). F1 Score, Accuracy, ROC AUC and PR AUC: Metrics for Models. Deepchecks.com. https://www.deepchecks.com/f1-score-accuracy-roc-auc-and-pr-auc-metrics-for-models/