Aerial Cactus Identification

AML Challenge 1
Group n° 50

Victor MAYAUD[1], Elliot BOUCHY[1], Aymeric LEGROS[1] and Naveen JOHNSON VALLAVANATT[1]

[1] Data Science, EURECOM, France

Abstract

This study explores deep learning approaches for columnar cactus identification from aerial imagery. We introduce a custom CNN architecture that employs a stacked convolutional block design. Each block consists of a Conv2D layer for feature extraction, followed by MaxPooling2D for spatial downsampling, BatchNormalization for regularization, and Dropout to prevent overfitting. Experiments are conducted on a dataset of 21,500 images from the VIGIA project. The custom CNN achieves a high accuracy of 99% and an ROC/AUC score of 1.0. Our findings demonstrate the effectiveness of deep learning techniques in accurately identifying columnar cacti, contributing to conservation efforts and ecological research. The proposed approach showcases the potential of AI in biodiversity preservation.

Index Terms: Deep Learning, Cactus, Convolutional Neural Networks (CNN), Transfer Learning, ROC/AUC score

1 Introduction

Quantifying the impact of human's activities on Earth's flora and fauna is anything but easy. Indeed, in Mexico for instance, logging, mining, or agriculture are so many activities that can negatively impact biodiversity without been well noticed.

Some supervisions are available determine either or not there are cactus, for example, but the aerial image may not always be clear which complicates the task. However, what a human cannot do for classification may be done with machine learning.

In this work we will use and discuss some machine learning techniques to accurately and perfectly determine either or not the flora on the picture is a Neobuxbaumia tetetzo cactus.

The report is led by the following plan: Section 2 briefly describes the dataset used in our stud. Section 3 presents and motivates the data preprocessing techniques we applied. Whereas section 4 illustrates the machine learning models selection we chose and how we selected it. Section 5 for its part focuses on the model evaluation and how we confirmed our choice. Finally, Section 6 finally concludes the report, summarizing our main findings.

2 Dataset

The "Aerial Cactus Identification" dataset was created as part of competitions and academic projects aimed at developing computer vision algorithms for the recognition of cacti in aerial images. The images were captured by a drone or other aerial devices, making them ideal for pattern recognition in natural and extensive environments.

2.1 Composition of the Dataset

The dataset consists mainly of two elements: Aerial images: These images are taken from a high vantage point, which allows to capture large areas of terrain.

Class labels: Each image is associated with a label that indicates whether or not a cactus is present in the image.

The typical data distribution in this dataset is typically about 74.9% of images with cacti and 25.1% of images without cactus.

3 Data Preprocessing

The data mining phase of the "Aerial Cactus Identification" dataset is of crucial importance for understanding the composition and distribution of the data. Initially, by examining the distribution of classes, we can see that the dataset is unbalanced, with 13,136 images containing 4364 cactus and not containing cactus. This distribution is visualized using a pie chart where cactus images represent the majority.

As for the characteristics of the images, they are usually pretreated before being used for model training. Images are resized to a standard size of 32x32 pixels and normalized for pixel values between 0 and 1.

3.1 Balancing

In order to correct the imbalance between the classes of the dataset "Aerial Cactus Identification", oversampling methods are implemented to increase the number of images of the minority class, that is to say those without cactus. These data augmentation techniques aim to balance class distribution and improve the ability of models to generalize on cactus-free images.

More specifically, these techniques include rotating images, adding noise, changing brightness, and randomly zooming. Rotation consists of rotating the images at predefined angles (90, 180 or 270 degrees), which adds variability to the training data. The addition of noise introduces random variations in image pixel values, simulating more diverse acquisition conditions. Changing the brightness makes it possible to create variations in lighting, making the models more robust to variations in brightness in real images. Finally, the random zoom performs random enlargements or reductions on the images, which allows capturing details at different scales.

By combining these methods, we obtain a more balanced and diversified training dataset, which allows models to better generalize on test data and improve their classification performance. This approach helps to reduce the risk of over-accurate models and to ensure a better ability to detect cacti in aerial images.

4 Methodology

After properly processing the dataset, the next critical step in our study is the development of a model capable of accurately detecting cactus within images. One of the most effective methods for image classification tasks is the Convolutional Neural Network (CNN). Renowned for their efficiency in handling image data, CNNs leverage powerful libraries that simplify implementation. By optimizing the architecture and carefully tuning the network's parameters, we can significantly enhance the model's accuracy and efficiency in classifying images. This chapter will delve into the detailed implementation of our CNN, followed by an in-depth exploration of the model's architecture and the rationale behind our specific design choices.

4.1 CNN Implementation

The implementation of the Convolutional Neural Network (CNN) utilizes the libraries TensorFlow and Keras. TensorFlow, a comprehensive Python library, facilitates the creation of complex machine learning models, while Keras, an open-source library, provides a user-friendly Python interface for artificial neural networks. To enhance the efficiency of the training process, we employed a GPU provided by Kaggle, which significantly accelerates computation.

Prior to training, the dataset was processed to ensure it was both balanced and normalized, optimizing the conditions for effective model training. The data was divided into two subsets: 90% allocated to the training dataset and the remaining 10% to the validation dataset. The training dataset undergoes rigorous training processes, while the validation dataset serves to evaluate the model's performance on unseen data.

To expedite the training phase, we set the batch size to 16 and planned for 50 epochs of training. During model compilation, we selected the Adam optimizer and binary cross-entropy as the loss function. The choice of Adam as an optimizer is due to its efficiency in handling sparse gradients and its adaptiveness in updating network weights, which is crucial for converging faster to the optimal weights. Binary cross-entropy is ideal for binary classification tasks, such as distinguishing between the presence and absence of cacti, as it measures the distance between the probability distribution of the output and the true distribution, providing a robust metric for model accuracy.

4.2 CNN model

This section focuses on the CNN architecture utilized for cactus detection, detailing the design choices and their justifications.

4.2.1 Detailed Architecture The model comprises multiple convolutional blocks, each designed to perform feature extraction at different levels of abstraction. Each block includes a convolutional layer followed by max pooling, batch normalization, and dropout, structured as follows:

- **Convolutional Layers:** Utilize filters of varying sizes to capture spatial hierarchies of features within the image. For instance, the initial layers use smaller filters to capture fine details, while deeper layers use larger filters to aggregate these details into higher-level features.
- **Max Pooling:** Reduces spatial dimensions, thus decreasing computational complexity and overfitting by summarily extracting dominant features that are robust against variations in the image.
- **Batch Normalization:** Normalizes the activations of the previous layer at each batch, maintaining mean output close to 0 and the output standard deviation close to 1. This stabilizes the learning process and dramatically reduces the number of training epochs required to train deep networks.
- **Dropout:** A regularization technique where randomly selected neurons are ignored during training, reducing the risk of overfitting. Our model employs a dropout rate of 20%, balancing between network complexity and training stability.

The architecture is terminated by a flatten layer transitioning to a dense layer which outputs the classification results. A sigmoid activation function in the dense layer makes it suitable for binary classification, indicating the presence or absence of a cactus.

Table 1
Detailed CNN Model Architecture for Cactus Detection

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 32, 32, 3)	0
batch_normalization	(None, 32, 32, 3)	12
conv2d	(None, 32, 32, 48)	3,648
max_pooling2d	(None, 16, 16, 48)	0
batch_normalization_1	(None, 16, 16, 48)	192
dropout	(None, 16, 16, 48)	0
conv2d_1	(None, 16, 16, 64)	76,864
max_pooling2d_1	(None, 8, 8, 64)	0
batch_normalization_2	(None, 8, 8, 64)	256
dropout_1	(None, 8, 8, 64)	0
conv2d_2	(None, 8, 8, 128)	204,928
max_pooling2d_2	(None, 4, 4, 128)	0
batch_normalization_3	(None, 4, 4, 128)	512
dropout_2	(None, 4, 4, 128)	0
conv2d_3	(None, 4, 4, 160)	512,160
max_pooling2d_3	(None, 2, 2, 160)	0
batch_normalization_4	(None, 2, 2, 160)	640
dropout_3	(None, 2, 2, 160)	0
conv2d_4	(None, 2, 2, 192)	768,192
max_pooling2d_4	(None, 1, 1, 192)	0
batch_normalization_5	(None, 1, 1, 192)	768
dropout_4	(None, 1, 1, 192)	0
flatten	(None, 192)	0
dense (sigmoïd)	(None, 1)	193

4.2.2 Architecture Choices The architecture was selected based on its success in similar image classification tasks as documented in recent literature. Modifications to standard architectures (e.g., adjusting filter sizes and the number of layers) were made to tailor the model to the specific challenges posed by our dataset, which includes a variety of cactus shapes and sizes under different lighting conditions.

4.2.3 Hyperparameters The selection of hyperparameters was a critical aspect of model tuning, aimed at optimizing both the model's performance and its computational efficiency. Each hyperparameter setting was carefully chosen based on its impact on the model's ability to generalize and learn from the training data effectively. Here we provide a detailed look at these choices:

- **Filter sizes and numbers**: The convolutional layers utilized varying filter sizes and numbers, strategically selected to extract and learn rich feature representations at various levels of abstraction. For instance, initial layers used smaller filters to capture fine details and edges, while deeper layers used larger numbers of filters to aggregate these features into more complex patterns. This approach helps in capturing sufficient contextual information without losing detail by being either too broad or too narrow.
- **Number of layers**: The architecture comprises several layers of convolution, pooling, and fully connected layers. The depth of the model, characterized by multiple

convolutional and pooling layers, was designed to be deep enough to capture complex features but balanced to avoid excessive computational burden. This ensures that the model can process higher-level features without becoming overly complex, which could lead to overfitting.

- **Trainable parameters**: A total of 1,567,175 trainable parameters were included in the model. These parameters are those that the model learns from the training data and adjusts through backpropagation. The high number of trainable parameters indicates the model's capacity to learn detailed and nuanced features from the data.

These hyperparameters are critical for shaping the model's learning structure and dynamics, directly influencing how well the model can learn and generalize from the training dataset to unseen data.

4.2.4 Optimization and Loss Function

Adam optimizer was chosen for its adaptive learning rate capabilities, which helps converge to the minimum more efficiently. Binary cross-entropy loss function was used due to its efficacy in binary classification problems, measuring how far off the predictions are from the actual classifications.

This model's configuration has proven effective, as evidenced by our validation results, which demonstrate robust accuracy and the model's ability to generalize well to unseen data.

5 Experiments

In this section, we detail the implementation of the proposed network and describe the used dataset. Then, we discuss the obtained results and examine the proposed model's performance.

5.1 Implementation Details

In this study, the experiments are carried out using a Kaggle notebook with the following configuration properties: an NVIDIA Tesla P100 GPU with 16GB memory; Intel Xeon CPU @ 2.30GHz with 4 cores; and 29GB RAM. The notebook is programmed using Python 3 and utilizes the TensorFlow and Keras libraries for building and training the convolutional neural network (CNN) model.

5.2 Balancing

After performing oversampling, we achieved an equal distribution of images containing cacti and images without cacti in the training set, as illustrated in Figure 1.

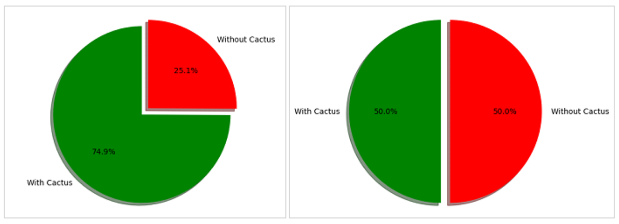

Figure 1. Distribution of train set before and after balancing

5.3 Results

To evaluate the proposed model's performance, precision, recall, Receiver Operating Characteristic (ROC) curve/Area Under the Curve (AUC), and loss metrics were utilized. The overall performance was assessed using a validation set comprising 1,750 images. As presented in Table 2, by the end of model training, the proposed network achieved an ROC/AUC score of 1.0 for both the training and validation data sets. Additionally, the model exhibited a loss of 1% on the training data and 0.6% on the validation data.

Table 2
Summarised Training History for 50 epochs

Epoch	Loss	Val Loss	ROC/AUC	Val ROC/AUC
0	0.184	0.112	0.979	0.993
10	0.044	0.027	0.998	1.000
20	0.030	0.056	0.999	1.000
30	0.021	0.017	0.999	0.999
40	0.018	0.005	1.000	1.000
50	0.017	0.006	0.999	1.000

To provide a detailed analysis of the model's performance during the training period, Figure 2 illustrates the Receiver Operating Characteristic (ROC) curve/Area Under the Curve (AUC) score.

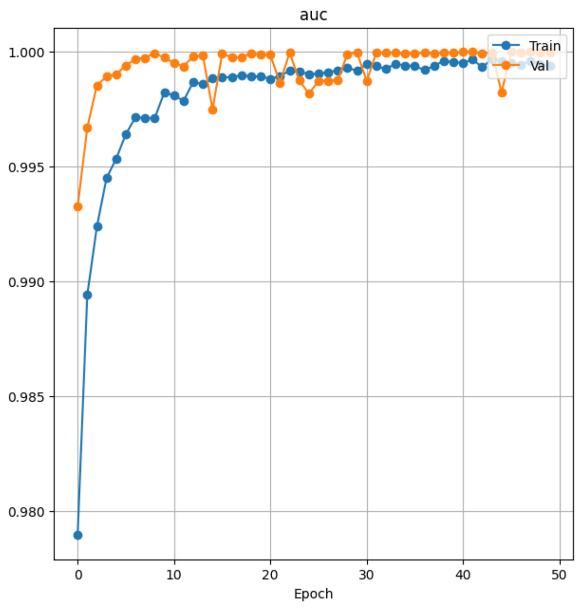

Figure 2. ROC/AUC Curve for Training and Validation Sets

For further evaluation of the model performance, Figure 3 illustrates the Binary Cross-Entropy Loss score.

To check whether the mode generalized well, 10% of the training was split randomly and results of confusion matrix for the model are shown in Figure 4.

Finally, in Figure 5 we ensure the model's accuracy by visually inspecting its predictions. Specifically, we continued sampling images until the model identified 3 images with cacti and 3 images without cacti. We then verify that the model's predictions aligned with the actual content of the images, demonstrating its effectiveness in the cactus recognition task.

6 Conclusion

In this work, we explored the application of machine learning for a classification problem by using a CNN.

Our work on the data with different steps such as preprocessing them and balancing them, were major steps in order to make our model efficient. This efficiency has been validate through various performance metrics [5.3].

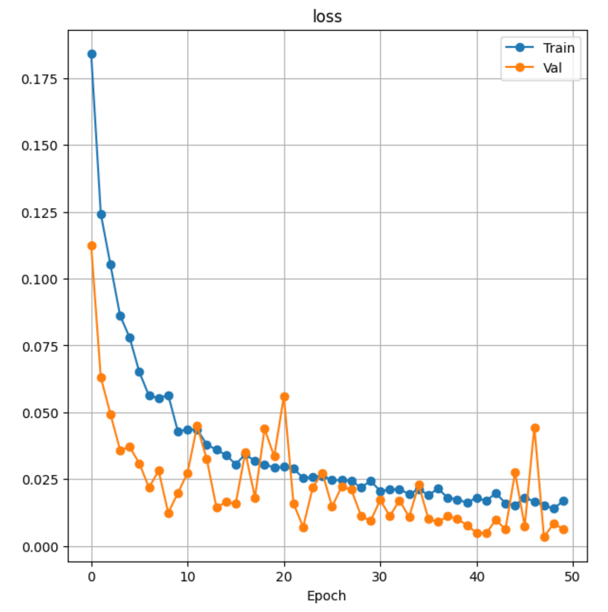

Figure 3. Binary Cross-Entropy Loss for Training and Validation Sets

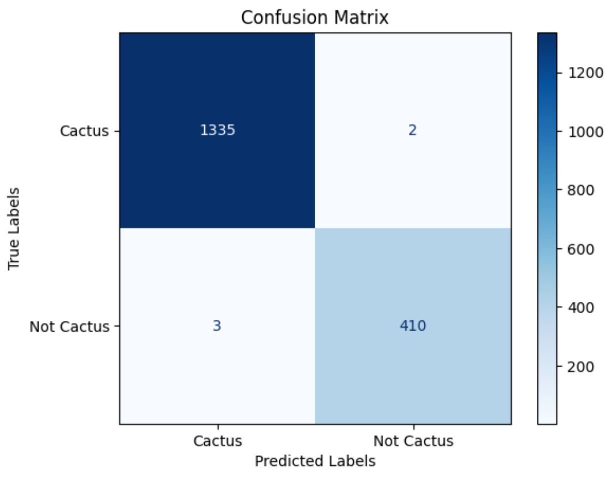

Figure 4. Confusion matrix for the validation set

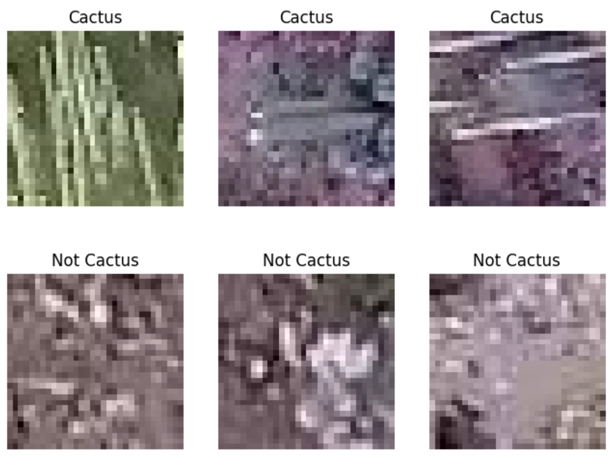

Figure 5. Model predictions for Cactus and Non Cactus

Finally, we are proud to present to you a 0.99 accurate machine learning model for the identification of Neobuxbaumia tetetzo cactus.

Acknowledgements

This research received support during the AML course, instructed by Professor Pietro MICHIARDI, Head of the Data Science Department at EURECOM, France.