Lab 1

AML

Elia Faure-Rolland
Tarek Saade
Ana Martinez
Jonas Thalmeier

30/04/2025

EURECOM
Sophia Antipolis

1 Introduction

In this project, we address the task of identifying the presence of a specific cactus species (*Neobuxbaumia tetetzo*) in aerial imagery. This challenge is motivated by the broader goal of supporting autonomous surveillance systems for protected natural areas, such as those developed under the VIGIA project in Mexico. Such systems play a crucial role in monitoring biodiversity and detecting human impact (e.g., through agriculture or logging) in ecologically sensitive regions. The dataset consists of 17,500 labeled images of size 32×32 pixels, each categorized as either containing a cactus (class 1) or not (class 0), resulting in a binary classification task. Among these, 13,136 images are labeled as containing a cactus, while only 4,364 images do not, revealing a substantial class imbalance.

Our objective was to develop an image classification model that achieves strong predictive performance while remaining lightweight in terms of memory and computational requirements. This allows for efficient training and inference, making the approach suitable for eventual deployment in embedded or resource-constrained settings. The combination of compact image size and imbalanced classes presented specific challenges that we addressed through careful data preprocessing and model design.

2 Preprocessing

The dataset was obtained from the corresponding Kaggle competition page, where it is provided in compressed format. After downloading and unzipping the files, we loaded the image labels from the accompanying `train.csv` file, which maps each image filename to a binary label indicating the presence of a cactus.

To ensure robust evaluation, the dataset was split into training, validation, and test subsets using a stratified sampling strategy that preserved the original class distribution in each split. Specifically, 80% of the data was used for training, and the remaining 20% was further split evenly into validation and test sets, resulting in an overall 80/10/10 split. This stratification was important to prevent data leakage and ensure fair performance evaluation across all subsets.

The raw image data and corresponding labels were then loaded into memory and stored as `numpy` arrays. This allowed for efficient manipulation and augmentation of the data prior to model training. Once preprocessing and augmentation were complete, the data was converted into PyTorch tensors and wrapped in a custom `Dataset` class. The class handled the formatting of input tensors to the shape $[N, 3, 32, 32]$ and ensured compatibility with PyTorch's data loading pipeline. Overall, the preprocessing pipeline was straightforward and followed standard practices for small-scale image classification tasks.

3 Handling Class Imbalance

Given the significant imbalance in the dataset—with class 1 (cactus) appearing approximately three times more often than class 0 (no cactus)—model training could be biased toward the majority class. One conventional approach to address this issue is to use class weighting in the loss function, assigning higher penalties to misclassifications of the minority class. While effective, this method introduces a dependency on hyperparameters that must be carefully tuned.

Instead, we chose a data-centric approach and applied data augmentation to balance the dataset. The augmentation process included horizontal and vertical flipping of the minority class images, which introduced spatial variation while preserving the semantic content of the image. By duplicating and augmenting the minority class, we constructed a balanced training set with an equal number of cactus and non-cactus images. This method allowed us to retain the benefits of a stratified validation and test set, while training the model on a dataset that did not exhibit class imbalance.

This approach had two key advantages. First, it avoided the need to introduce class weighting or sampling strategies that can complicate training dynamics. Second, it introduced additional diversity into the training data, which can improve generalization and model robustness. The augmented dataset was subsequently converted into PyTorch tensors, ready for training using the chosen model architecture.

We applied data augmentation techniques exclusively to the training set, preserving the original class imbalance in the validation and test sets. As a result, accuracy alone may provide a misleading indication of model performance, since a naive classifier that consistently predicts the majority class could still achieve a high accuracy score. To address this, we evaluated our model using the F1-score, which balances precision and recall. This metric provides a more informative assessment by penalizing both false positives and false negatives, making it particularly well-suited for imbalanced binary classification tasks such as this one.

4 Model Architecture

Given that the task involves binary image classification, we selected a Convolutional Neural Network as our model architecture, which can be seen in Figure 1. CNNs are particularly effective for processing image data, as they are designed to capture spatial hierarchies and local patterns. The input images in our dataset are low-resolution (32×32 pixels), which informed our choice to build a lightweight architecture that avoids excessive depth or complexity. The network, implemented in PyTorch, consists of two convolutional layers with ReLU activations to extract local features, followed by max-pooling operations to downsample the feature maps and reduce dimensionality. These are followed by a fully connected hidden layer that integrates the learned features. Dropout regularization is applied before the final dense layer to reduce the risk of overfitting.
Since we used `BCEWithLogitsLoss` for training, which internally applies a sigmoid activation to convert logits into probabilities, we did not include a separate sigmoid function in the model architecture.
The architecture was intentionally kept simple to reduce computational overhead and training time while maintaining sufficient expressiveness to learn the relevant patterns. This design choice not only enables fast experimentation and tuning but also ensures compatibility with deployment in resource-constrained environments.

5 Experiments and Results

During training, we used the *Adam* optimizer and the `BCEWithLogitsLoss` as our loss function. Adam is widely adopted in deep learning for its adaptive learning rate mechanism, often leading to faster and more stable convergence compared to traditional optimizers like stochastic gradient descent (SGD), especially in small to moderately sized neural networks such as ours. `BCEWithLogitsLoss` is specifically designed for binary classification tasks and is well-suited for models that output raw scores (logits). By combining the sigmoid activation with the binary cross-entropy loss in a single function, it improves numerical stability and simplifies the training pipeline.

In the initial phase of our experiments, we focused on determining an appropriate number of training epochs.

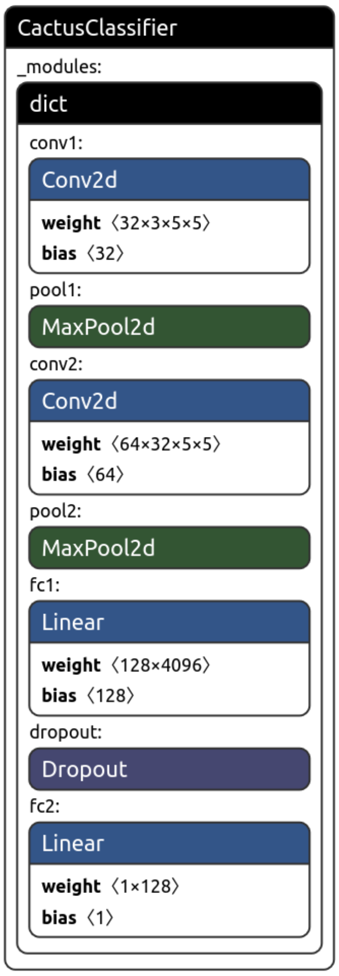

Figure 1: Model Architecture.

To do so, we trained the model for 200 epochs using standard hyperparameters and analyzed the learning curves. As shown in Figures 2 and 3 , the validation curves for accuracy, F1-score, and loss begin to diverge from the training curves around epoch 40, indicating the onset of overfitting. Based on this observation, we decided to truncate training to 40 epochs in subsequent experiments to prevent overfitting while allowing sufficient learning.

We then performed a two-stage hyperparameter tuning strategy to optimize the model's performance. In the first stage, we explored architectural hyperparameters by conducting a grid search over convolutional kernel sizes {3, 5, 7, 9} and dropout rates {0.1, 0.3, 0.5, 0.7}. Each configuration was trained for 40 epochs on the balanced training dataset and evaluated using validation accuracy and F1-score, as shown in Figure 4. The results revealed that a kernel size of 5 consistently yielded the best performance, likely offering a favorable trade-off between receptive field size and model complexity. Larger kernels (e.g., 9×9) increased memory consumption and slowed training without improving accuracy. Dropout rates in the range of 0.1–0.3 were most effective, while higher dropout values like 0.7 significantly degraded performance and increased instability in the validation curves. The best configuration from this stage used a kernel size of 5 and a dropout rate of 0.1.

In the second stage, we fixed the architectural parameters and tuned optimization-related hyperparameters by testing learning rates {0.1, 0.01, 0.001, 0.0001} and batch sizes {32, 64, 128}. To further stabilize training, we evaluated several values for weight decay and found that 1×10^{-5} reduced validation oscillations without causing underfitting. All experiments were tracked using

Weights & Biases, allowing for effective monitoring and comparison.
The best results were achieved using a learning rate of 0.001, a batch size of 128, and a weight decay of 1×10^{-5}, yielding the highest validation accuracy and F1 score. The final model, trained with these optimized hyperparameters, demonstrated strong generalization performance on the test set, achieving a validation F1 score of 0.9852 and an accuracy of 0.978.

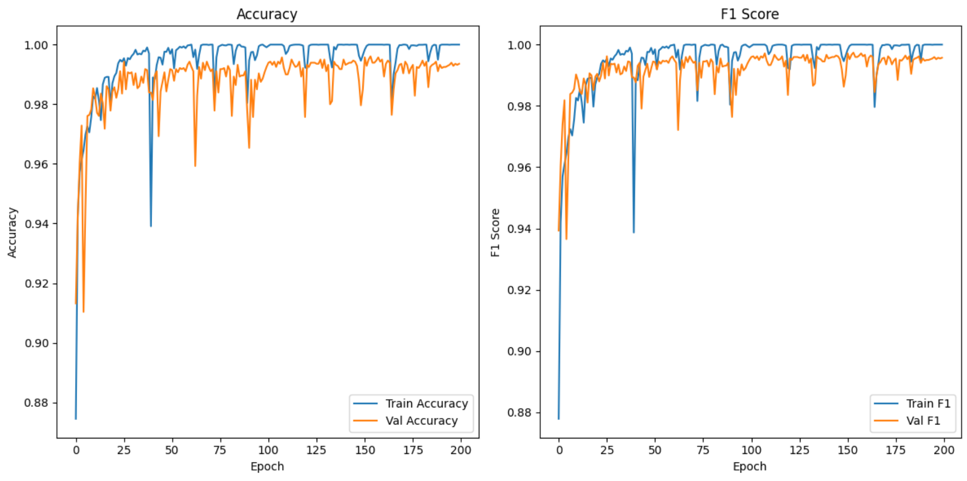

Figure 2: Accuracy and F1 score for Validation and Training sets.

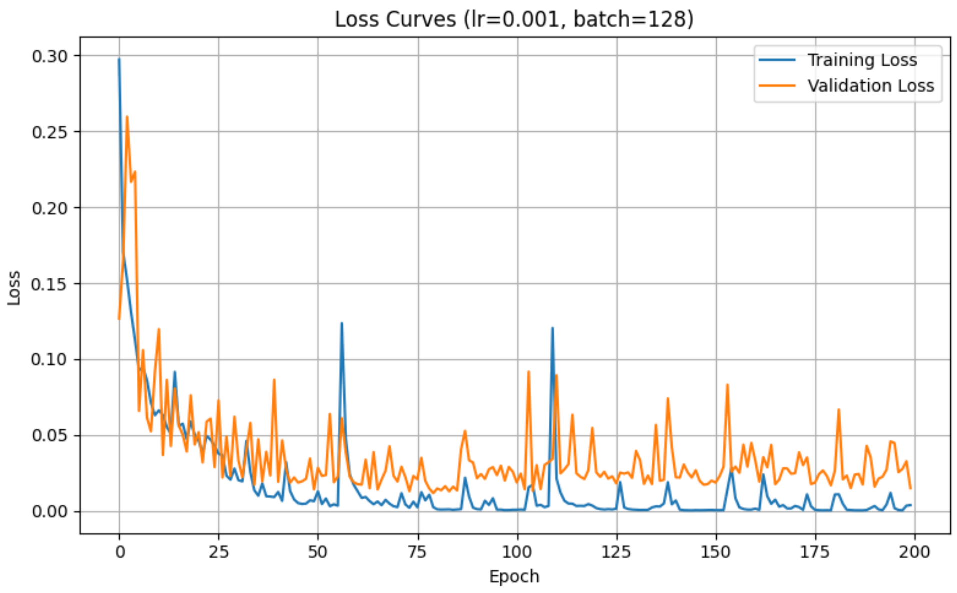

Figure 3: Loss curve for Validation and Training sets

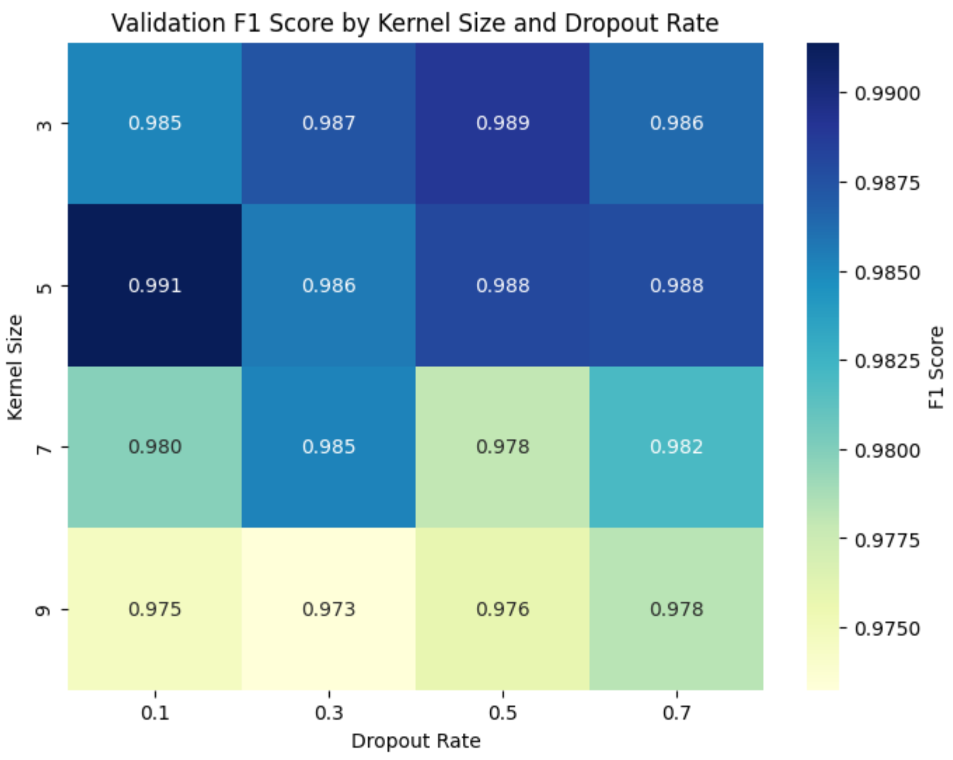

Figure 4: Validation F1 Score by Kernel Size and Dropout Rate.

6 Conclusion

In this project, we successfully developed an efficient and accurate image classification model for detecting the presence of *Neobuxbaumia tetetzo cacti* in aerial imagery. Our approach addressed key challenges such as low-resolution inputs, class imbalance, and limited computational resources—constraints commonly encountered in real-world ecological monitoring applications. By employing a lightweight convolutional neural network architecture and using data augmentation to balance the dataset, we were able to improve the model's robustness and fairness without relying on complex reweighting schemes.

Through systematic hyperparameter tuning, we identified optimal settings for kernel size, dropout rate, learning rate, and batch size, achieving a strong validation F1 score of 0.9852. These results confirm the effectiveness of our design and training strategy, demonstrating that high performance can be attained even with compact models tailored for deployment on embedded or resource-constrained devices.

Given the already excellent performance, fast inference, and low parameter count of our initial model, we decided not to explore more complex architectures. We deemed further improvements unlikely and prioritized simplicity and efficiency over marginal gains, aligning with the practical requirements of ecological monitoring in the field.