Aerial Cactus Classification

LEONE, Andrea
Politecnico di Torino
Institut EURECOM
leone@eurecom.fr

BONINO, Giulia
Politecnico di Torino
Institut EURECOM
bonino@eurecom.fr

NEPOTE, Luca
Politecnico di Torino
Institut EURECOM
nepote@eurecom.fr

May 2, 2024

Abstract

This study addresses the task of aerial cactus identification, which is part of an environment preservation project in Mexico. Using a dataset composed of low resolution images of cacti and local flora, we aim at correctly classifying the images as containing a cactus or not. We experiment with different CNN-based models and propose a Custom convnet with 3 convolutional layers and data augmentation which achieves the best performance.

1 Introduction

The purpose of this project is to create a model that is able to determine whether an image contains a cactus or not. The dataset used for this objective was proposed by Mexican reasercher in the VIGIA (Vigilancia Autónoma de Reservas de la Biósfera) project [4], whose main aim was to automatize the surveillance of protected areas, which have been endangered due to human activities.

2 Data analysis

Out training dataset is composed of 17500 images captured from aerial point of view. All the pictures are of the same size: 32×32 pixels. The dataset contains two classes: one with images of cacti and one without. From Figure 1 we can observe that the dataset is unbalanced, i.e. it contains about three times as many pictures with cacti as the number of pictures without cacti.

We were also given an unlabeled test set, which is composed of 4000 images. We investigated the distribution of the training and test set after normalization and scaling in Figure 2 where we plotted the distributions of the two datasets. From the image we can notice that the two distributions are mostly super-imposed, but they have different peaks, which we think may be due to a different percentage of images with cacti in the test set.

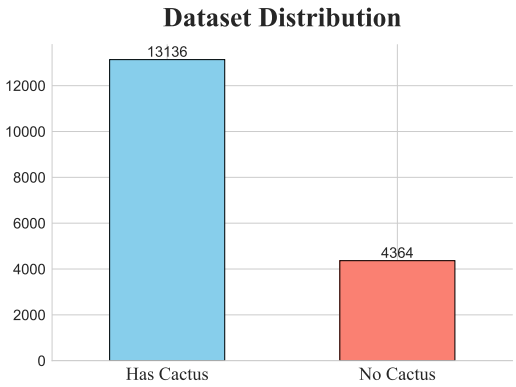

Figure 1: Dataset Distribution

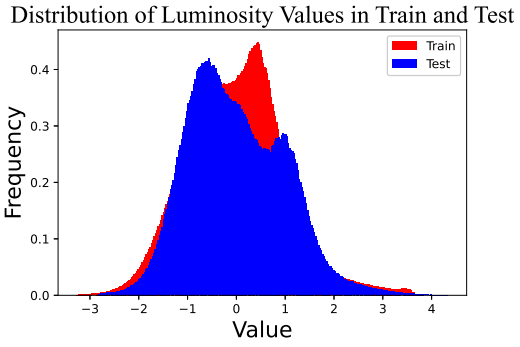

Figure 2: Distribution of luminosity for train and test sets

3 Data processing

As described in Section 4.3, we implemented some experiments using transfer learning: to be precise, we used the ResNet18 model [2] pretrained on ImageNet [1]. Therefore, in order to finetune the model with our dataset, we resized all the images to 256×256 pixels and then did a center crop to 224×224 to match the correct size of the pictures in ImageNet. Moreover, we normalized and

scaled the images with the mean and standard deviation of ImageNet.
Instead, for the other models we tried, we simply normalized and scaled the images in order to have zero mean and unitary variance.
For every model we did an 80-20 split (train + validation) of the training dataset, to do both hyperparameters tuning and model selection. We didn't take a split for the test set because, as we saw in Figure 2, the distribution of the given test set seems to be different from our labeled data. Therefore if we reported performances on a test set taken from our training dataset, they would probably not be consistent with performances on the actual test set.

Finally, to address the unbalance between the two classes, we performed **Data Augmentation**: we cloned the images which were part of the minority class (i.e. the no-cactus class) and performed some transformations on them, such as random flip with probability 0.5 and rotation with an angle of degrees extracted at random.

We employed data augmentation only on the Custom CNN, which had the best performances among the models we experimented with.

4 Model selection

In this section we are going to analyze different model choices that we tested for our purpose and we will select the best one. We will start with a baseline and then we will analyze more sophisticated solutions.
For evaluating the models we employed the following metrics:

- *Validation loss* (we also computed *Training loss*): the chosen loss function is the *Binary Cross Entropy*. Even though this is generally employed for binary classification tasks with balanced datasets, we noticed that it was performing as other losses more suited for unbalanced datasets, like the *Focal* loss. Furthermore, we will deal with class-unbalance by using other metrics and data augmentation.
- *Error rate*;
- *F1-score*: expressing the balance between precision and recall, this metric is often used when dealing with unbalanced datasets.

About neural networks: all neural networks have been trained with a batch size of 64 and for a maximum of 100 epochs with an early stop condition.

4.1 Support Vector Machine

Description: As a baseline model, we implemented a simple SVM with RBF kernel. To account for the unbalanced dataset, we set the parameter *class_weights = 'balanced'*, which multiplies the regularization parameter C with a weight which is inversely proportional to class frequency. We performed a GridSearch for the hyperparameters using these values:

- C: 0.1, 1, 10
- gamma: 0.001, 0.01, 0.1

Data Preparation: the images have been normalized and converted to grayscale. We also performed histogram equalization.

Line detection and Issues: As the shape of the species of cacti in our dataset was mostly straight and vertical, we wanted to extract features that highlighted the presence of straight lines, therefore we aimed at using the Hough transform. In order to achieve this, we experimented with different methods to improve the images: we used Gaussian and Median blurring to remove noise, then we performed the Canny edge detection algorithm, removed isolated edges and finally applied the Hough transform to detect lines.
However, despite our efforts, we encountered a problem: the Hough transform struggled to detect the lines in the images. We think that our dataset's resolution was too low and the lines present in the images were not found due to the noise.

Results: the Grid Search found that the best parameters were C=1 and gamma=0.001, yelding the results reported in Table 1.

tpr	fpr	error rate(%)	F1-score
0.8743	0.1716	13.71	0.9053

Table 1: SVM results

With the next models we expect to obtain much better performances with respect to these.

4.2 Custom CNN

Description: the first model we tried, excluding the baseline, was a custom CNN: given the low resolution of the dataset images, we thought that this could be sufficient to achieve good results.
Our custom CNN is made of 3 convolutional layers for feature extraction and 2 fully connected layers with dropout for the actual classification. The dropout percentage and the width of the fully connected layer (which we will refer to as *FCsize*) have been tuned with a grid search. The described model is shown in Figure 3 (generated with Netron [3]).

Data preparation: to make the model more robust, the data has been standardized as explained in Section 3.

Validation process: the first thing we did was trying three different learning rates (10^{-4}, 10^{-3}, 10^{-2}) to pick

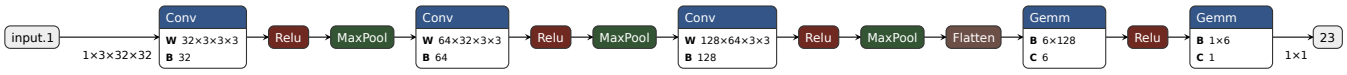

Figure 3: Custom CNN

the one which best balances convergence speed with a fine estimate. We picked the value 10^{-3} as the best one, and we will use it for all neural networks.
Then we proceeded with a grid search for the following hyperparameters:

- dropout: 0.1, 0.3, 0.5
- FCsize: 6, 12, 24

We picked the best model from the grid search and tested it with different seeds to check if the results were stable. We also did a comparison with the second best model.

Results: the grid search gave us the validation losses (top value) and F1-scores (bottom value) shown in Table 2.

	FCsize		
Dropout	*6*	*12*	*24*
0.1	0.0279 0.9949	0.0266 0.9932	0.0256 0.9946
0.3	**0.0194** **0.9959**	0.0299 0.9948	0.0253 0.9934
0.5	0.0263 0.9944	0.0252 0.9949	0.0281 0.9940

Table 2: Grid search - Custom CNN

The optimal value in terms of validation loss is highlighted, and is the one with FCsize = 6 and dropout = 0.3. Full statistics about the optimal model are shown in Figure 4.

Even though we considered validation loss for picking

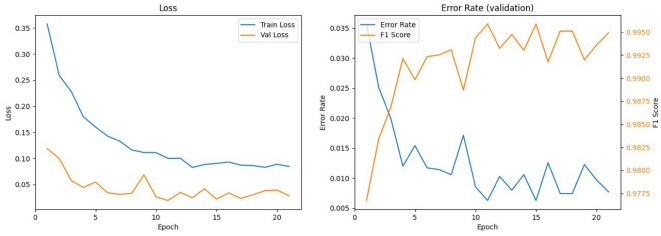

Figure 4: Best custom CNN results

the best model, we also took into account the error rate and F1-score. In this case the selected model is the best also in terms of F1-score, which is very close to 1, meaning that we achieved a nearly perfect balance of precision and recall.

The error rate obtained with such model is 0.62%.
By retraining this model with 10 different seeds, we obtained the following average statistics:

train loss	val loss	error rate(%)	F1-score
0.0675	0.0372	1.00	0.9933

Table 3: Avg results

The validation loss increased significantly and also other significant metrics like the F1-score got worse, but much less in proportion. This means that the seed that we had selected previously was "lucky", based also on the fact that the validation error ranged between 0.0231 and 0.0883 among the different seeds, and the F1-score ranged between 0.9903 and 0.9953.
To have an additional criteria to decide which model is the best, we repeated the training with multiple seeds also with the second best model from the grid search (FCsize = 12, dropout = 0.5), obtaining the following average results:

train loss	val loss	error rate(%)	F1-score
0.0477	0.0344	1.05	0.9930

Table 4: Avg results 2nd

The average validation loss of the second best model is actually better, but the error rate and especially the F1-score are roughly the same. With this in mind, we still think that the model with FCsize = 6 and dropout = 0.3 can still be taken as an optimal one, since it is has fewer parameters than the 2nd best and the average results are more or less the same.

4.3 ResNet18 fine-tuning

Description: after seeing the very good performances of the previous model, we decided to try to change the feature extraction part of the custom CNN with something already well-known. We opted for ResNet18 for its very good reputation with image classification tasks. We left the last two fully connected layers of the custom CNN the same, but changed all the previous part with ResNet18. Since this model will take much longer to train and test, we also decided to fine-tune just the FCsize, fixing the dropout to 0.3 which resulted to be the optimal value from the experiments with the custom CNN.

Data preparation: the input data has undergone standard preprocessing, aligning it with the typical transformation applied when utilizing a pretrained network trained on the ImageNet dataset (see Section 3).

Validation Process: we fine-tuned the FCsize, trying the values 1, 6, 12 and 24.

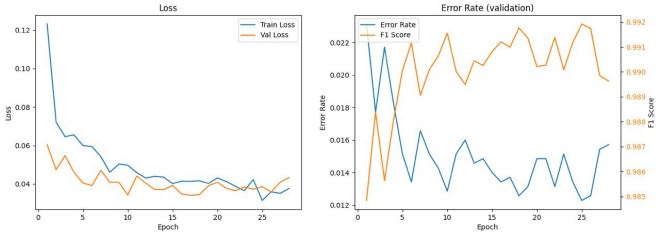

Figure 5: Best ResNet18 results

Results: the best model turns out to be the one with FCsize = 24 (Figure 5), obtaining the results in Table 5 after 17 epochs of training.

train loss	val loss	error rate(%)	F1-score
0.0414	0.0341	1.37	0.9910

Table 5: Best ResNet18 results

The obtained performances are worse (or at most similar) with respect to the custom CNN, so we stopped here the tests with this model because we didn't expect to get any better results and the cost in terms of computational resources and time to train such model is much larger than the cost for the custom CNN.
We think that, as a further experiment, one could try to use a bigger ResNet model (e.g. ResNet34 or better), but it would be like shooting a fly with a cannon given the simplicity of this task and the great performances already obtained by the custom CNN.

4.4 Custom CNN + Data Agumentation

Description: At this point, considering that the best model so far is the Custom CNN, we decided to train it again using also data augmentation, to see if it's possible to get any improvement.

Data preparation: we did the same preprocessing as in Section 4.2 but we also added data augmentation, as explained in Section 3.

Validation process: since this model is relatively light to train and test, we decided to perform the same grid search as in Section 4.2.

Results:
Based on Table 6 we should pick the model with FCsize = 24 and dropout = 0.1 as the best one. However, if we have a look at the loss vs. epochs graphs of these models,

	FCsize		
Dropout	*6*	*12*	*24*
0.1	0.0074 0.9975	0.0102 0.9964	**0.0057** **0.9985**
0.3	0.0082 0.9972	0.0062 0.9974	0.0097 0.9996
0.5	0.0074 0.9983	0.0107 0.9962	0.0071 0.9981

Table 6: Grid search - Custom CNN + Data Aug.

it turns out that all those with FCsize = 24 have a very volatile validation loss curve (Figure 6), which makes us think that these model have poor training stability and might overfit the training data. This also happens with models having FCsize = 12.

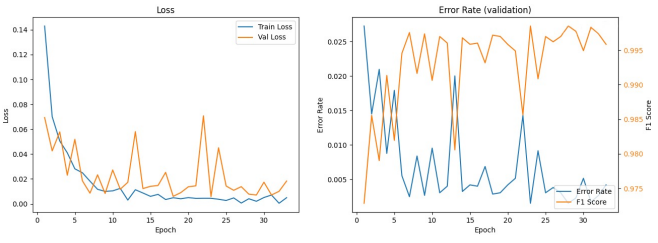

Figure 6: Custom CNN + Data Aug. FCsize=24, dropout=0.1

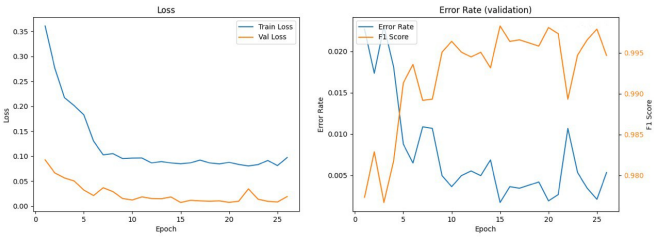

Figure 7: Custom CNN + Data Aug. FCsize=6, dropout=0.5

On the other hand, models having FCsize = 6 have a very smooth (Figure 7) validation loss curve, which means that they are more stable, so we are going to pick the model with FCsize = 6 and dropout = 0.5 as the *candidate* one for this section. We prefer this one over the one with dropout = 0.1 because a higher dropout means that the loss curve is smoother, making the model more stable. Also notice how the validation loss is just a bit higher w.r.t. the model having FCsize = 24 and dropout = 0.1, but the F1-score is basically the same. The results in the table were achieved after 15 epochs for our candidate model and 23 epochs for the model with FCsize = 24 and dropout = 0.1.

The error rate of the candidate model (FCsize = 6, dropout = 0.5) is 0.17%.

5 Test

From the previous section, we stated that the optimal model is the custom CNN with data augmentation and FCsize = 6 and dropout = 0.5. To finish with our experiments, we decided to pick 200 images at random from the unlabeled test dataset, classify them with such model, and then verify the prediction by hand. We were 3 people doing a majority voting for hand labeling the images.
With this process we were in disagreement with the model for 7 images out of 200, so we could say that the approximate error rate of the model is 3.5%. However, we should also keep in mind that the images have a very low resolution and it's often very hard to understand if there is a cactus or not also for a human eye. The following image (Figure 8) shows two predicted images. The left one is correctly classified while the second one is wrongly classified, in our opinion.

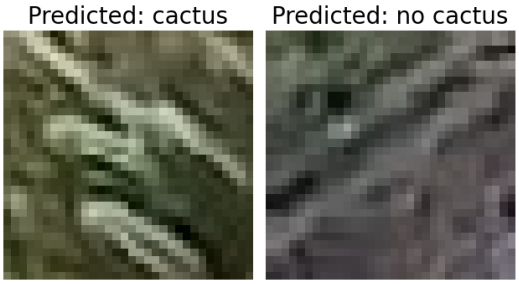

Figure 8: Tested images

Overall, we think that our model is generalizing quite well.

6 Conclusions

In this project we dealt with a classification problem concerning aerial images, in order to predict whether there was a cactus or not. For this purpose we first implemented an SVM as baseline model; then we tried more complex models, such as a Custom Convnet architecture and a fine tuned model based on ResNet18: from the results it can be seen how using a simple model based only on three convolutional layers with Relu activation functions and Batch normalization it is possible to obtain very good results. This is due to the fact that we are considering small and simple images, for which also a basic model is enough for the classification task.
We also tried to experiment with image pre-processing techniques (e.g. Hough transform) without success, but we were able to improve the custom CNN performances by means of data augmentation techniques.

In conclusion, we suggest to use the custom CNN with an FCsize of 6 and a dropout percentage of 50% trained with augmented dataset for this classification task. This solution seems to be the optimal in terms of validation loss, accuracy and F1-score. It is also a very light DeepNet (94'029 parameters for a total size of $\tilde{0}$.36MB) with respect to other solutions, which makes it possible to implement on devices with limited resources.

References

[1] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Lutz Roeder. Netron. URL: `https://netron.app`.

[4] Juan Irving Vasquez. Vigia, 2016. URL: `https://jivg.org/research-projects/vigia/`.