AML - Challenge 1: Aerial Imagery

Francesco Dente, Michele Ferrero, Niccolò Zanieri

April 2024

1 Introduction

To assess the impact of climate change on Earth's flora and fauna, it is vital to quantify how human activities such as logging, mining, and agriculture are impacting our protected natural areas. Researchers in Mexico have created the VIGIA project, which aims to build a system for autonomous surveillance of protected areas. A first step in such an effort is the ability to recognize the vegetation inside the protected areas.

In this study, we explore different machine learning techniques in order to address a binary classification task to distinguish between aerial images that contain cacti and those that do not contain cacti.

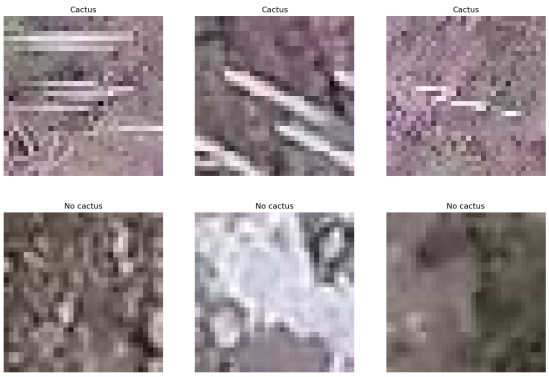


Figure 1: Cacti photos

2 Data Analysis

2.1 Dataset

The dataset contains a large number (17500) of 32 x 32 thumbnails of aerial photographs of a columnar cactus (Neobuxbaumia tetetzo), together with the corresponding label (*has_cactus*).

As we can see from Figure 2, the dataset is imbalanced towards the cactus class, thus we may need to employ different techniques to try and address this problem.

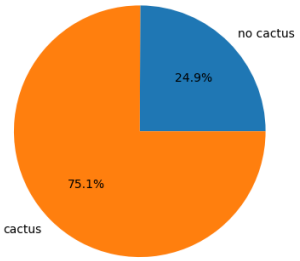


Figure 2: Dataset classes distribution

2.2 Data Preprocessing

We also need to be aware that we are dealing with image data, so we want to choose our preprocessing techniques accordingly. For non-convolutional approaches, we may want to manually extract significant features from our image data before feeding them into our model, leading to more complex preprocessing techniques. On the other hand, using convolutional networks for feature learning will lead to the use of much less complex techniques.

2.2.1 Scaling

For each experiment, we rescale our image data pixels to a predefined range [0,1], since this is the default applied by the pytorch transform ToTensor(). This a very common practice when dealing with image data in order to treat all the images in the same manner and avoid that higher pixel range images result in higher loss than lower pixel range images.

2.2.2 Data Augmentation

We exploit data augmentation for two different aims:

Regularization: for each experiment we randomly flip horizontally images at training time.

Oversampling: we can use it to increase the number of minority class samples to address the problem of our unbalanced dataset.

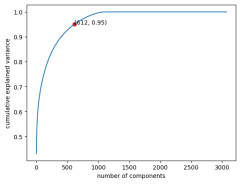
2.2.3 Standardization

We normalize data to have a standard deviation of 1 and a mean of 0.

Standardization helps to improve the quality and consistency of the data. What we do for our dataset is standardization for each channel, based on the mean and variance of all pixels of the training set only, to avoid **data leakage** in validation and testing.

2.2.4 Principal Component Analysis

We can also look at PCA since we are dealing with images and we expect many features (pixels) to have no variance. This will be helpful for non-convolutional models to reduce the number of irrelevant features.



3 Model selection

Now we will analyze the performance of different machine learning models for our task, using **AUC** as an evaluation metric to finally choose our optimal model. In this way, we can evaluate how our models are generally able to separate classes, regardless of the chosen classification threshold, using a metric that is generally insensitive to class imbalance.

We will explore three different approaches:

- **Non deep approach**: we will use a simple **Logistic Regression** as a baseline
- **CNN**: we will use a CNN (**LeNet5**) that we will train from scratch
- **PreTrained CNN with Transfer Learning**: we will use **ResNet** as features extractor and then train the last fully connected layers

Since we get almost perfect results, especially when using CNNs, we will also explore the **memory usage** and **inference time** as interesting metrics for choosing our model.

3.1 Training procedure

We split our dataset into train-validation-test (70%, 20%, 10%) maintaining the same balance between classes as in the starting data (75/25). We use the validation set to check for overfitting during our training phase, exploiting **early stopping** in case the AUC keeps increasing on the training set but not on the validation.

Then, we retrain our chosen model on both train and validation and we evaluate on the test set to see how well our model generalized.

We use PyTorch **BCEWithLogitsLoss** function for training our models. This loss combines a Sigmoid layer and the **BCELoss** in one single class. This is more numerically stable than using a plain Sigmoid followed by a **BCELoss** as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.

3.2 Logistic Regression

We first start by training a simple Logistic Regression baseline model and then try different preprocessing techniques to see which ones yield the best results. The most interesting ones will then be used for the other two approaches.

We report results for the best combination of preprocessing techniques [1].

As expected, PCA proves to be helpful (since we use a non-convolutional model), as does standardization. On the contrary, oversampling does not help significantly, so we will not use this technique for the other two networks.

A simple logistic regression with basic preprocessing techniques (PCA and standardization) can solve our task quite well (0.946 AUC), suggesting that the dataset is almost linearly separable. With more sophisticated handcrafted features and this simple model we could probably get very high scores; we decided not to investigate further as CNNs will automatically extract features for us.

[1] We employed a grid search in order to find them

Preprocessing	AUC on validation set
-	0.907
PCA-612	0.942
Oversampling PCA-612 Standardization	0.940
PCA-612 Standardization	0.946

3.3 LeNet-5

The second model we decided to use is LeNet-5 [1] whose original architecture is shown in Figure 3. In order to adapt the network to a binary classification program we turned the output layer from a 10 elements layer into a single element one.
We chose to use LeNet-5 to evaluate the performance of a network smaller than ResNet34 but still able to exploit automatic feature extraction.
In this case we decided to train two different networks, one without any pre-processing and then the other one applying the following transformations: random horizontal and vertical flip at each epoch, standardization and scaling.
This procedure shows how using the latter configuration leads to an improvement in both average loss and **AUC** metric over unseen data.

3.4 ResNet34

We then trained a model using ResNet34, which required some manipulations. First, we had to resize our 32x32 images to fit the input size of ResNet34 (224x224), and then we had to take a slightly different approach to the training phase than for the other models.

To deal with the binary classification task, we changed the fully connected layers (head) of the model from a simple layer to:

Layer	Input	Output
Linear	512	256
ReLu	-	-
Dropout(0.5)	-	-
Linear	256	1

The training phase is then performed in two training cycles:

- **First cycle**: we freeze the convolutional parameters and we train only the last fully connected layers
- **Second cycle**: we unfreeze all the parameters and we fine-tune the whole network

We obtain a perfect **AUC** of 1 on the validation set regardless of the preprocessing techniques, showing the great power of such a pretrained network used as a feature extractor. However, this comes at the cost of higher memory consumption and higher inference time.

3.5 Model choice

We now compare our two best performing convolutional models[2] taking into account AUC, memory usage and inference time as metrics.

We see that LeNet5 is more lightweight than ResNet34 but still obtains a score close to perfection on both the validation and test sets.

For this reason, we believe that LeNet5 would be a better choice, allowing to perform inference in a faster and less hardware demanding way [3].

Model	AUC val	AUC test	Params	Infer. time
LeNet5	0.996	0.995	$\sim 60K$	$\sim 5ms$
ResNet34	1.000	1.000	$\sim 21.4M$	$\sim 100ms$
LR	0.946	0.950	613	$\sim 100\mu s$

[2]The results are obtained using scaling and normalization as preprocessing techniques
[3]Benchmarks performed on a laptop with i7-9750H CPU @ 2.60GHz × 12 and GeForce RTX 2060 Mobile 6GB

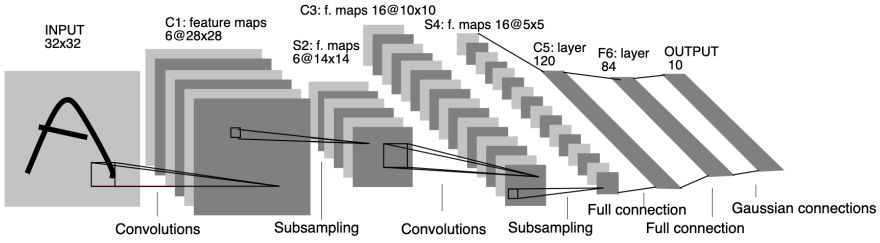


Figure 3: Original LeNet5 architecture

4 Unsupervised approach to increase model performance

Lastly, we tried to make use of the unlabeled data we had to increase our best model performance. Our goal was to roughly label it in order to use it as additional training data for our LeNet5 network.

We explored two different methods, and we used the accuracy on our **known** validation set as evaluation metric.

Model	Accuracy on our validation set
Unsupervised clustering methods (KMeans)	0.474
Zero-Shot LLM (CLIP)	0.249

Looking at the very poor results obtained by these two unsupervised methods, we decided to use our heavy ResNet34 network, which performs perfectly on the known validation set. In this way, we are able to improve our lightweight LeNet5 network by exploiting the strength of the ResNet34 model. We obtain an AUC of 0.999 on the validation set and 0.997 on the test set. The results are very similar to those obtained before, but since we trained our model with more data, we expect it to generalize better to unseen data.

5 Conclusions

We explored three different approaches to the problem of cacti detection in aerial images, comparing classical machine learning approaches and modern deep learning techniques. We also focused on evaluating differences in widely used architectures in relation to the problem at hand. The obtained results clearly show how small networks can give very promising results and it is not necessary to always use excessively big architectures. It is noteworthy to stress how even very simple manipulation of the training dataset can lead to better performance capabilities of our model. We also consider of high importance how using a less accurate but simpler network could allow to get more than acceptable results and to run inference on low resources devices.

References

[1] LeCun Y, Bottou L, Bengio Y, Haffner P. Gradient-based learning applied to document recognition. Proc IEEE. 1998

[2] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun Deep Residual Learning for Image Recognition Microsoft Research 2016