2024 AML Challenge 1 Group 20

Moaad MAAROUFI Barthelemy CHARLIER

EURECOM

Abstract

In the context of the Aerial Cactus Identification Kaggle competition, this work aims to investigate a set of approaches to this problem. It begins with some exploratory data analysis and image processing techniques. We experiment with simple models, as well as complex and less traditional ones.

1 Introduction

The Aerial Cactus Identification competition on Kaggle is part of the VIGIA project, which aims to build a system for autonomous surveillance of protected areas. In the competition, we are tasked with creating an algorithm that can identify a specific type of cactus in aerial imagery.

For this purpose, we explore a set of machine learning and image processing techniques, motivate their use in this context, and show their strengths and limitations. These techniques include edge detection, denoising, morphological opening, logistic regression, SVM with RBF kernel, CNNs, VGG pretrained on ImageNet as part of a transfer learning model, and a less traditional approach, a variational autoencoder for anomaly detection.

2 Explatoratory Data Analysis

The data is composed of a training and test dataset with 32x32x3 RGB images. The training set is labeled, while the test set is not. For this reason, we will only use the training set in our study. The training set contains 17,500 images belonging to two classes: cactus or no cactus. It is quite imbalanced, with 2/3 of the dataset belonging to the cactus class.

There is some diversity in the images, with the cactus not always in the same place and sometimes rotated. This diversity is great as it will push the models to generalize. However, note that the images are noisy, which will give a hard time to some models, as we will see.

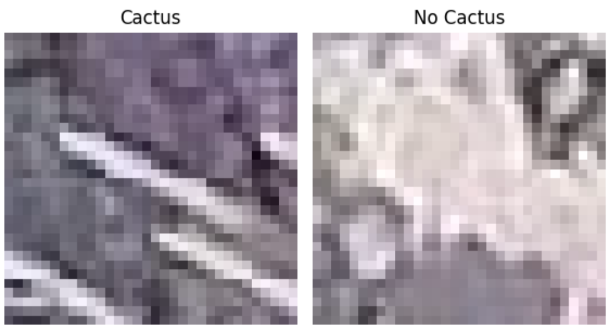

Figure 1: Cactus vs no Cactus

3 Data preprocessing

The data was cleaned and massaged in numerous ways, but not all the transformations made it into the final pipeline. Nonetheless, we felt it would be useful to cite what worked and what didn't. At each step in the model selection section, we will explain the results.

As we mentioned before, the data is very imbalanced. In the training/validation/testing split, we have to take this into account because the worst-case scenario is that during the split, most of the No_cactus (minority class) ends up in validation and testing. For this reason, we use a stratified split based on the label, which allows us to have the same proportion of each class across the splits (70% train/15% valid and 15% test).

Depending on the model, the images were either normalized using a standard scaler (for shallow learning models.) or with 0.5 mean and 0.5 variance across all channels: $normalizedvalue = (original\_value - 0.5)/0.5$ (for deep models). This is especially beneficial for pretrained models on ImageNet, where the images have a mean of $[0.485, 0.456, 0.406]$ and a standard deviation of $[0.229, 0.224, 0.225]$, thus it is a good idea to try to match this distribution.

Furthermore, to counter the imbalanced nature of the dataset, we augment it by applying some transformations (random rotation/flipping) to the minority class and adding the new images into the training set until both classes have the same cardinality.

We also experimented with other image processing techniques, such as grayscale to reduce the dimensionality for the smaller models, this way the dimensionality is divided by 3 (RGB previously). We also tried denoising, edge detection, and morphological opening to connect the detected edges. These transformations were applied to the entire dataset, not for augmentation, as they change the images' dimensionality. Note that the models used in our study require the inputs to have the same dimensionality.

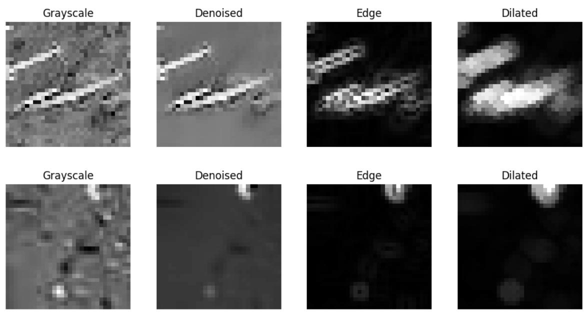

Figure 2: Cactus vs no Cactus

4 Model evaluation

We are dealing with a binary classification problem. Therefore, for training the models, we will use binary cross-entropy as the loss function.

The dataset is skewed, so precision alone wouldn't be a good metric. Since a model that always predicts cactus would score better than random chance by definition, we will also rely on recall and the F1 score to judge the performance of a model, in addition to precision.

5 Model selection

5.1 Logistic regression

As a baseline approach, even though we have images as input, it's a good idea to see how a linear model such as logistic regression would perform. This will give us an idea of what the more complex models should surpass.

The idea here is to provide the logistic regression model with data that has been normalized using a standard scaler and flattened.

After this step, we noticed that we already had great performance, but the model was underperforming on the minority class. To address this, we augmented the dataset with more examples of this class, using only horizontal flipping and random rotation by 10 to 40 degrees. These transformations were chosen because they wouldn't harm the training process. However, the results weren't as we had hoped; the performance of the model worsened in both classes, even with increased regularization (L2).

This could be because the model is simply memorizing the training data and failing to generalize. The images of non-cactus plants are not a distinct class of their own, as they are quite different from each other. The only thing they have in common is that they do not contain a cactus. As a result, the model may be overfitting to the training data and failing to learn more generalizable features.

For the third attempt, we used the data that went through the image processing techniques (grayscale, denoising, edge detection, and morphological opening). This approach also reduced the performance of the model. This could be attributed to the reduction in the number of parameters of the model (which, for logistic regression, is equal to the dimensionality). In this case, the dimensionality was divided by 3 because of the grayscale conversion. This reduction in parameters may have limited the model's ability to learn more complex patterns in the data.

Data	**Class**	**Precision**	**Recall**	**F1-score**
Original	Class 0	0.77	0.85	0.81
	Class 1	0.95	0.92	0.93
Augmented	Class 0	0.61	0.09	0.15
	Class 1	0.76	0.98	0.86
ImProc	Class 0	0.70	0.78	0.74
	Class 1	0.93	0.89	0.91

5.2 SVM with RBF

The same argument that motivates the use of logistic regression also applies to this approach. However, the purpose of using an SVM with an RBF kernel is that the decision boundary may not be linear, so it makes sense to consider a model with a non-linear decision boundary.

All the experiments that have been done for logistic regression have also been conducted for this model, including data preprocessing, augmentation, and hyperparameter tuning.

The SVM with RBF kernel performs relatively better than logistic regression, suggesting that the decision boundary may indeed be non-linear. The same remarks about the performance of the model that were made for logistic regression still apply here. However, the data augmentation with flipping and rotation to counter the imbalance in the dataset worked better for this model. We achieved better recall for the non-cactus class, and the performance on the cactus class did not degrade as much as it did with logistic regression.

Data	**Class**	**Precision**	**Recall**	**F1-score**
Original	Class 0	0.90	0.84	0.87
	Class 1	0.95	0.97	0.96
Augmented	Class 0	0.87	0.87	0.87
	Class 1	0.96	0.96	0.96
ImProc	Class 0	0.79	0.73	0.76
	Class 1	0.91	0.93	0.92

6 CNN

This is an image classification problem, so a robust model that is expected to perform well is a convolutional neural network (CNN). The automatic feature extraction of CNNs makes them particularly powerful for this kind of task.

For the architecture design, we followed a set of principles and conventions to come up with the design:

- Use a low number of channels at the beginning (to detect low-level features) and then increase gradually the number of channels as the architecture gets deeper (to combine and detect high-level complex features).

- General filter sizes : 3x3, 5x5 and 7x7. Max-Pooling parameters: 2x2 or 3x3 filter sizes with a stride of 2.
- Get inspired from classic networks like LeNet, AlexNet, VGG-16, and VGG-19 because they work.

Following these principles, we started with two architectures as a baseline: Conv-Pool-Conv-Pool and Conv-Conv-Pool-Conv-Conv-Pool, with a number of channels 16-32-64-128 or 16-16-32-32, respectively. We also included some dropout and batch normalization in the linear layers. We used a batch size of 64 for 30 epochs.

Note that the models are checkpointed based on the validation error. We use the best model for testing. With these baseline models, we achieved near-perfect results on both classes, with an F1 score of 0.99 for class 1 and 0.98 for class 0.

We noticed that the validation loss for the second architecture (Conv-Conv-Pool-Conv-Conv-Pool) was a bit spiky and unstable. This instability was evident when we retrained the models on an augmented dataset with flipped and rotated images, just like in the previous section. The second model diverged during training, while the first model (Conv-Pool-Conv-Pool) showed further improvement in performance.

This website helped us design the architectures by giving us the size of the activation at each stage.

Model	**Dataset**	**Class**	**Precision**	**Recall**
Model 1	Original	0	0.96	0.99
		1	1.00	0.99
	Augmented	0	0.98	0.99
		1	1.00	0.99
Model 2	Original	0	0.98	0.98
		1	0.99	0.99
	Augmented	0	0.43	1.00
		1	1.00	0.55

7 VGG 16

At this stage, we have significantly outperformed the baseline. Nevertheless, we want to explore pre-trained models to see how they would perform on our task.

Transfer learning is a technique used in image classification where a pre-trained model is used as a starting point to build a new model for a different but related task. The pre-trained model has already learned features from a large dataset, such as ImageNet, and these features can be reused for the new task.

VGG16 is a popular pre-trained model that has been shown to perform very well on the MNIST dataset, which is comparable in size to ours (28x28). This motivated our choice to use VGG16 as a starting point for our model.

We fine-tuned the pre-trained VGG16 model by freezing the feature extraction layers, removing the base classifiers, and adding a custom one. We then trained the model on the normalized dataset (normalized in the same way as ImageNet, as discussed in the data processing section). Our goal was to achieve optimal score with the least number of training epochs possible.

Flatten	-
Dense	256 units
Activation	'relu'
Dropout	0.5
Dense	1 unit
Activation	'sigmoid'

Table 1: Classifier architecture added to VGG16 network

The results were very promising, with minimal training the model achieves nearly perfect scores:

Class	**Precision**	**Recall**	**F1-score**
0	0.98	0.98	0.98
1	0.99	0.99	0.99

The following table shows how the model perfomrs with different hyperparameters:

Epochs	**LR**	**Batch**	**F1 Avg**
1	1E-06	32	0.7876
2	1E-06	32	0.4287
3	1E-06	32	0.9831
1	1E-06	64	0.9709
2	1E-06	64	0.9470
3	1E-06	64	0.9760
1	1E-05	32	0.9797
2	1E-05	32	0.9852
3	1E-05	32	0.9081
1	1E-05	64	0.9505
2	1E-05	64	0.9903
3	1E-05	64	0.9778

Table 2: Hyperparameters vs avg F1 score of the two classes

8 Convolutional Variationnal autoencoder

This problem can be seen from another angle: anomaly detection where the images with no cactus represent the anomaly. For this purpose we will use a variational autoencoder.

A Convolutional Variational Autoencoder (CVAE) transforms the input using the encoder(compression with CNNs) into a compact latent vector, z. The decoder(upsampling with transposed CNNs) then attempts to reconstruct the original input from z. The model is trained by minimizing the reconstruction error and the KL divergence between z's distribution and a standard normal distribution, which serves as a regularization term in the loss. Loss = Reconstruction Error + KL Divergence.

The architecture of this model was inspired by another problem of activity detection(soccer or not soccer), which also involved smaller images (60x80x3). There was a need to adjust the input dimensions along with fine-tuning the latent dimensionality to better suit the current task.

In our case, the model was trained on the cactus class only (which solves the imbalance problem as we do not use the minority class at all here). It is expected that in the validation phase, where the model is given both classes, the reconstruction error of the no_cactus class will be much larger than that of the cactus class. This difference in reconstruction error will allow us to classify images as cactus or no cactus.

Below is a plot of the reconstructed images from the test set after training. (We fixed 12 normal vectors from the latent space, from which we sample to plot the images for reproducible results).

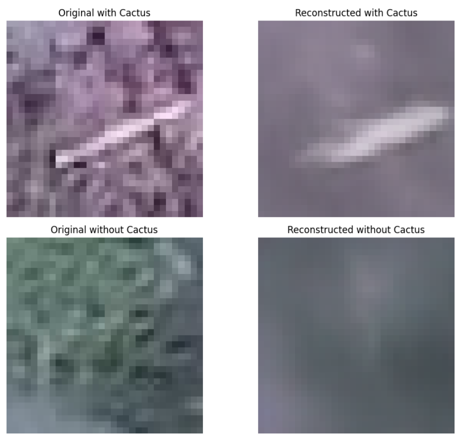

Figure 3: Original vs reconstructed images

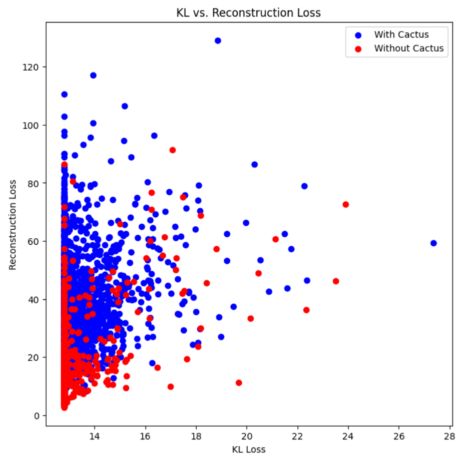

Figure 4: Kl divergence vs recontrcution error

Class	Precision	Recall	F1-score
0	0.84	0.75	0.79
1	0.65	0.76	0.70

The results from the first experiment were not as expected. After training the model for over 100 epochs with checkpointing (reconstruction loss on the validation set), the reconstruction error of the non-cactus images was lower during testing, which was surprisingly odd since the model was trained to reconstruct cactus images. Upon investigation, we found that non-cactus images are easier to reconstruct due to containing fewer features. Additionally, the similarity in style between the two classes (the background of cactus images is non-cactus images) could be a reason.

We also conducted some other experiments, such as making the architecture deeper by adding more layers, using the denoised images from the auto-encoder to train the model instead of the original dataset, in hopes of reducing the effect of noise and finally considering the cactus images as the anomaly. However, these experiments yielded similar or worse results compared to our previous approaches.

9 Training Platform and Reproducibility

We used Google Colab (T4 GPU) as our main working platform, but encountered issues with slow training and data processing times. We found that Google was throttling the bandwidth in Google Drive, where we were reading data and checkpointing our models.

Initially, we were reading images one by one from disk and applying data preprocessing steps, but this was too slow due to random reads hitting the disk 17,500 times(size of the dataset), taking 40 minutes to finish training for CNNs. To solve this issue, we preprocessed the images locally on our machines and created a list containing the preprocessed images. We then pickled the list and loaded it into Google Drive, reading it back as a list object in Google Colab, which considerably sped up the training time to under 5 minutes for most of our models.

For reproducibility, we used a random seed for the train split and ran our models a minimum of three times, obtaining relatively consistent results for all of our models.

10 Conclusion

In this study, we experimented with various machine learning and data processing techniques, explaining their suitability and limitations. We conclude that the CNN model is likely the best option for computational efficiency and high performance. The VGG16 model may also be a good choice as it might generalize better in the case of a out-of-distribution (OOD) data, given that it was trained on ImageNet. However, this would require empirical verification. To test this assertion, we propose labeling the images in the testing set using an image-based question-answering transformer as an extension to our work. This approach will help verify the model's robustness to OOD data.

References

1. Efficient CNN tuning
2. In-depth-guide-to-convolutional-neural-networks
3. Activity-Detection-using-Variational-Autoencoders
4. Aerial-cactus-identification-vgg16-cnn