ADVANCED DATA SCIENCE REPORT

DELOIRE ALEXANDRE
JORGE RÉMI
HE JIAYI

2024-2025

SUMMARY

01

INTRODUCTION

Introduction

Characterizing land-use from aerial imagery is one of the key challenges in remote sensing. As land-use changes rapidly, the acquisition of a large volume of aerial images allows us to track these shifts effectively. However, due to the vast amount of data, manual analysis quickly becomes impractical. Consequently, many studies have focused on automating this task using deep learning methods, particularly convolutional neural networks (CNNs), which have proven to be powerful tools in the field of image recognition.

This report explores the effectiveness of deep learning models for characterizing aerial images, based on two distinct datasets: a single-label dataset and a multi-label dataset, both derived from the UCM Dataset. The first dataset consists of 21 land-use classes, such as "agricultural," "beach," or "tennis court," where each image belongs to a single class. The second is an extended version of the first, re-labeled with 17 classes in a multi-label setting, where an image can contain multiple objects or landscapes, such as "airplane," "tree," "sea," etc.

The goal of this project is to study, design, and evaluate automatic classification methods for these aerial images. The report is structured into six parts. We will begin with a theoretical introduction to convolutional neural networks (CNNs), followed by an exploration of specific models like ResNet for single-label classification. Next, we will present the results obtained, comparing the performance of different models tested, before addressing the challenges of multi-label classification. Finally, the conclusion will discuss the strengths and limitations of the approaches developed and outline potential areas for improvement.

02

SINGLE LABEL

Context

In this first part of the project, we focus on the single-label classification task using the UCM Dataset. This dataset consists of satellite images representing various land-use categories. Each image is associated with a single class among 21 predefined categories, such as "agricultural", "airport", "beach", "forest", and others. The main objective is to design a model based on Convolutional Neural Networks (CNN) that can accurately classify these aerial images according to land usage.

A Convolutional Neural Network is particularly well-suited for image recognition tasks due to its ability to capture complex spatial and visual features through filters applied to input images. These filters allow the model to detect patterns such as edges, textures, or shapes, which is crucial for distinguishing different landscapes in the context of satellite imagery.

We began by implementing a simple baseline CNN model that will serve as a foundation for further optimizations. In this section, we will detail the architecture of this initial model, evaluate its performance, and explore the impact of various hyperparameters, such as kernel size, the use of Batch Normalization, Dropout regularization, and the addition of extra convolutional layers. We will also analyze how image transformations (such as rotation, zooming, or flipping) applied to the training, validation and test datasets can enhance the model’s robustness and performance.

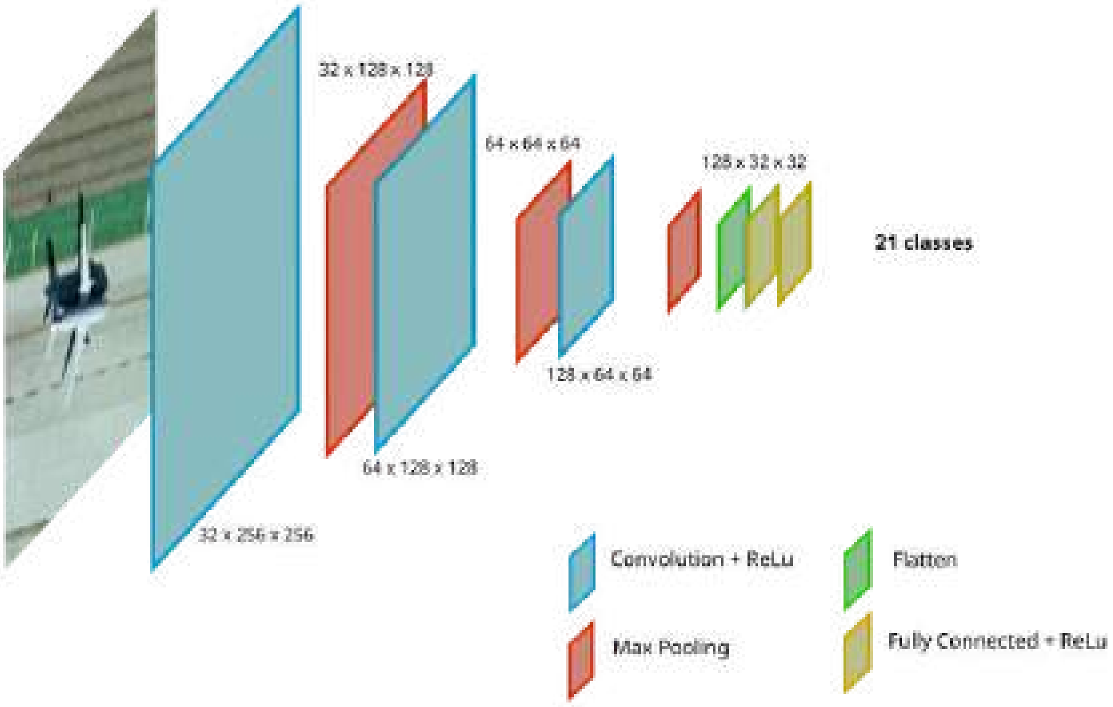
First Model

Our first model is a simple convolutional network consisting of three convolutional layers followed by max-pooling layers and ending with fully connected layers for classification. This model was designed to capture the spatial features of the images while allowing effective classification across the 21 classes of the dataset.

Convolutional layers are critical in CNNs for extracting local patterns from the image, such as edges or textures. Each filter applied in a convolutional layer helps detect specific features, which are then enhanced or transformed as the information passes through the different layers of the network. Max-pooling layers are used to reduce the dimensionality of the extracted features while preserving the most important information, thus speeding up the training process and minimizing the risk of overfitting.

Architecture

- Input: Images of size 256x256 with three channels (red, green, blue - RGB).
- Convolution 1: 32 filters of size 3x3, followed by a ReLU (Rectified Linear Unit) activation function and a MaxPooling layer (2x2). This first layer helps detect basic features such as edges.
- Convolution 2: 64 filters of size 3x3, followed by a ReLU activation and a MaxPooling layer (2x2). This second layer extracts more complex features such as patterns or textures.
- Convolution 3: 128 filters of size 3x3, followed by a ReLU activation and a MaxPooling layer (2x2). This layer detects even more abstract and detailed features.
- Fully Connected Layers: After flattening the extracted features, a dense layer with 512 units and a ReLU activation is used to capture the interactions between features. This is followed by a final classification layer with 21 outputs (one for each class), using a softmax function to generate the probability distribution for each class.



Architecture of the Initial CNN model

This basic architecture provides the foundation for relatively accurate image classification, though further optimizations are necessary to improve its performance.

Training Results

In this section, we present the results obtained from training the model. The performance on the test set will be detailed in the **results** section. The model was trained on 1470 images, with 420 images reserved for testing and 210 images used for validation. Training was carried out over several epochs until the performance on the validation set stabilized.

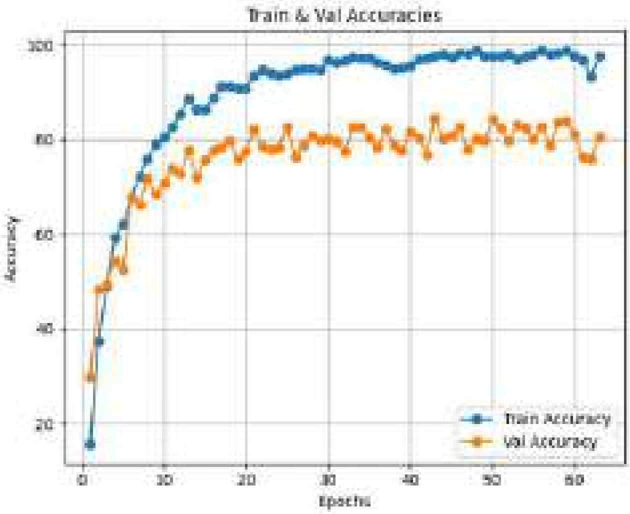
We used the Adam optimizer (Adaptive Moment Estimation) for training, as it is widely favored for its ability to adjust the learning rates of individual parameters dynamically. The learning rate was set to 0.0005, which allowed the model to converge slowly and avoid drastic oscillations in the network's weights. The training results are summarized below:

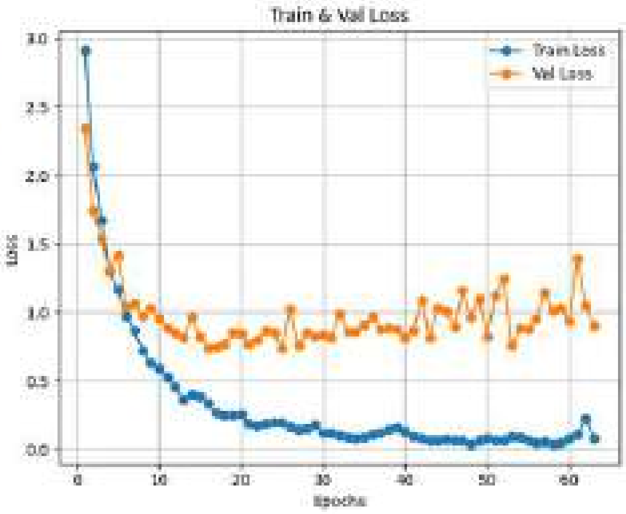
---------------- Training Results ------------------
Final Training Accuracy: 97.69%
Final Validation Accuracy: 80.71%
Final Training Loss: 0.0814
Final Validation Loss: 0.9041

Best Validation Accuracy: 84.52% at epoch 43

The validation accuracy reached a respectable score of nearly 85%. However, there is a notable gap between the training accuracy (98%) and the validation accuracy (81%), suggesting possible overfitting. This indicates that while the model performs exceptionally well on the training data, it struggles to generalize on unseen data.

Below, we present the results over 60 training iterations.





The graph on the left illustrates the model's accuracy (in blue) compared to the validation accuracy (in orange). We observe that training accuracy stabilizes around iteration 30, suggesting that training could have been stopped earlier to avoid unnecessary computation.

The graph on the right shows the loss functions for both training and validation. While the training loss consistently decreases, the validation loss reaches a minimum between iterations 20 and 30 before slightly increasing. This rise in validation loss could be a sign of overfitting, where the model starts memorizing the training data rather than generalizing effectively.

Impact of Parameters

The initial model is not perfect. When building a Convolutional Neural Network (CNN), it is crucial to adapt the different layers to the specific task at hand. Each layer in a CNN has a distinct role, and its effect on the overall model's performance often depends on its interaction with other layers. Therefore, testing various configurations and observing what works well and what doesn't is essential. Through numerous trials and adjustments of hyperparameters and architectural choices, we can enhance the model and achieve better test results.

In this section, we highlight some of the strategies that helped us better understand the role of each layer and the interactions between them. For example, filter size (kernels), the depth of the layers (number of filters), and the use of techniques like Batch Normalization or Dropout can have a significant impact on both model accuracy and generalization.

We tested many different combinations, sometimes with success and other times without. These experiments allowed us to better grasp the trade-offs between the model's ability to fit the training data and its robustness to new data (avoiding overfitting). A portion of this research is documented in the notebook accompanying this report, while we showcase only a small part of it here.

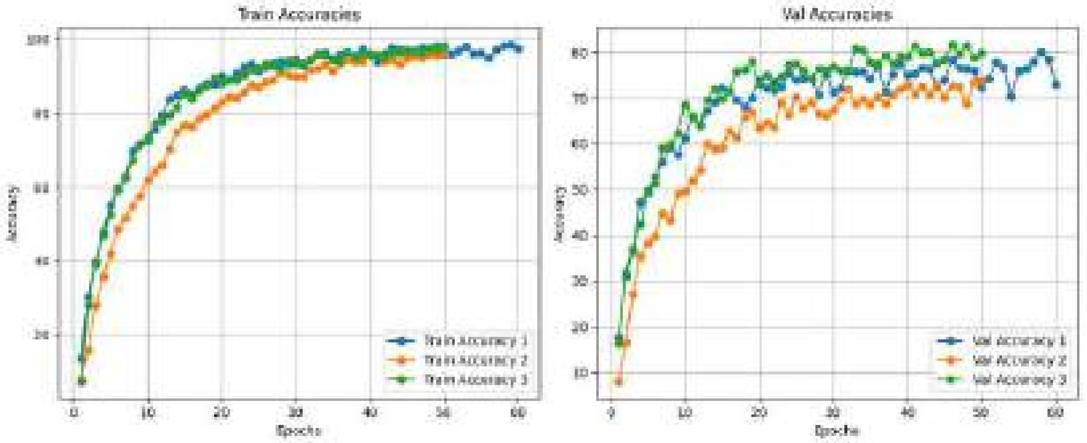
Undoubtedly, this was the most time-consuming aspect of the project, but it was also the most intellectually rewarding, as it deepened our understanding of CNNs. This experimental approach enabled us to gain a clearer insight into the internal workings of convolutional networks and their adaptability to complex tasks like image classification.

Number of layers

The number of layers refers to the number of levels in the convolutional neural network model. In our initial model, we currently have 3 layers, each composed of a convolutional layer followed by a max-pooling layer. These layers help to extract visual features at different levels of granularity, with earlier layers detecting simple patterns (such as edges) and deeper layers capturing more complex patterns.

To evaluate the impact of the number of layers on performance, we tested two other configurations: a model with 2 layers and a more complex model with 4 layers. Both were trained on the same test and validation sets as our initial 3-layer model.

The results are presented in the graphs below:



Legend:

Orange = 2 layers

Blue = 3 layers

Green = 4 layers

Based on these results, we observe that the difference in performance between the 3-layer and 4-layer models is minimal, though the 4-layer model seems to slightly outperform in validation. However, the 2-layer model clearly underperforms, showing significantly lower accuracy compared to the 3-layer model.

It is important to note that adding more layers increases the model’s complexity, which in turn leads to longer training times. Therefore, while a model with more layers may offer slight improvements in accuracy, it is essential to strike a balance between performance and the computational resources required. Small optimizations, such as adjusting the number of layers, must be carefully considered based on the available resources and specific project goals.

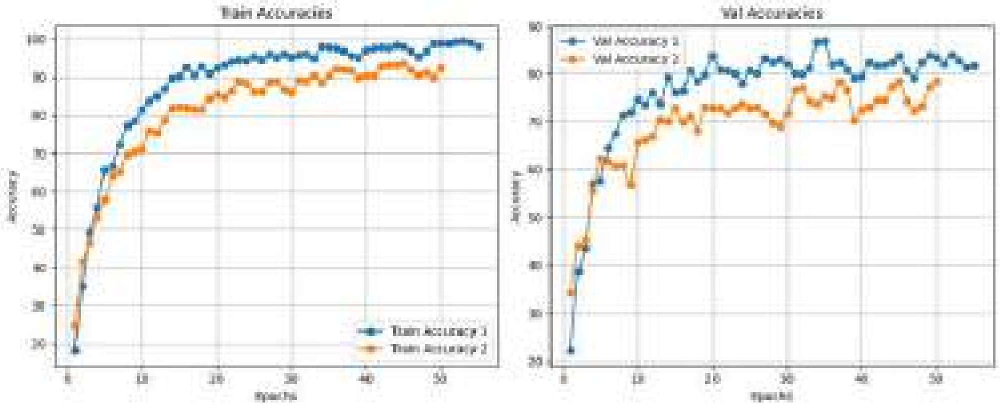
Number of channels

The number of channels in a convolutional layer refers to the number of filters used in that layer, with each filter capturing specific features of the image. In our initial model, we use 32 channels for the first layer, 64 for the second, and 128 for the third. We tested two alternative configurations: a lighter model with 16, 32, and 64 channels, and a heavier model with 64, 128, and 256 channels.

After training these two models, the lighter model showed performance nearly identical to the initial model, which was surprising given its lower complexity. Conversely, the heavier model underperformed by about 5% in validation, possibly due to overfitting or difficulty generalizing. Additionally, the heavier model took longer to train, making it less practical. As a result, we decided to stick with the current configuration of 32, 64, and 128 channels.

Kernel Size

The kernel size refers to the size of the filter applied to perform the convolution on an image. In most CNN models, we use a 3x3 filter, which is a common choice in the literature due to its effective balance between capturing local details and managing model complexity.



Legend:

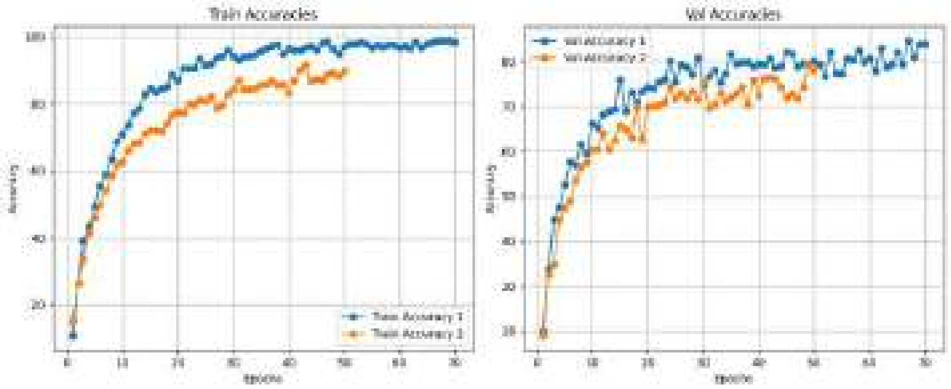
Orange = kernel 5x5

Blue = kernel 3x3

In this study, we also experimented with a 5x5 filter (with a padding of 2 to maintain consistent feature map sizes). The results showed that, while effective, this model slightly underperformed compared to the initial model with a 3x3 filter. We also tested larger filters (7x7) and asymmetric filters like 1x3 or 3x1. However, in all cases, the initial model with the 3x3 kernel proved to be the most efficient, confirming it as the optimal choice for this task.

Batch Normalization

Batch Normalization is a technique often used after the ReLU activation to stabilize and improve the training of neural networks by normalizing intermediate outputs from a layer. CNNs often benefit from this technique as it helps avoid large weight oscillations during training.



Legend:

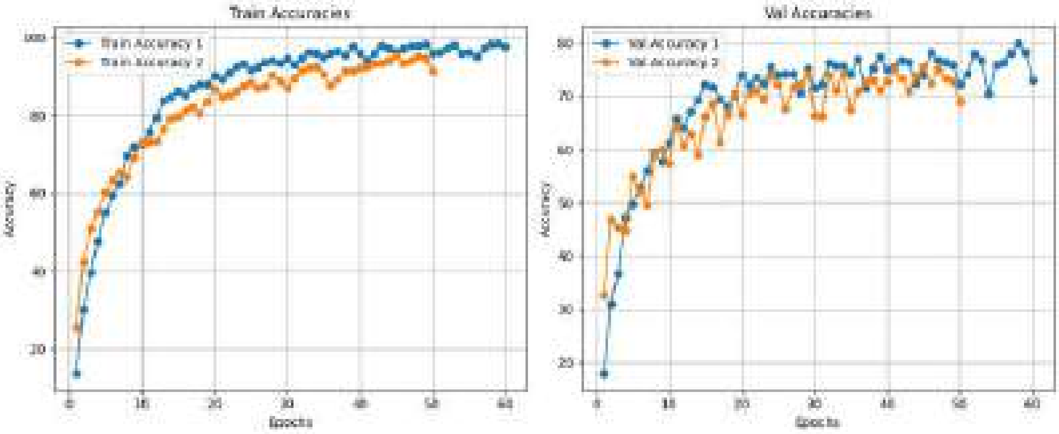
Orange = with Normalization

Blue = without Normalization

We applied Batch Normalization to our model to observe its impact. At first glance, the results appeared slightly worse than those of the initial model. However, this difference was primarily due to slower training. Batch Normalization tends to slow down model convergence, and since we were limited by time and resources, we couldn't extend the training for enough epochs. Increasing the learning rate did not yield better results either. Although we didn't include BatchNorm in our final model for performance reasons, it remains a crucial factor for more complex CNN architectures.

ReLu

A ReLU layer is straightforward in its operation: it sets all negative values to 0. This non-linear operation is quick to compute and adds minimal extra computational time to the model. We tested removing this layer to observe its effect on the model's learning process.



Legend:

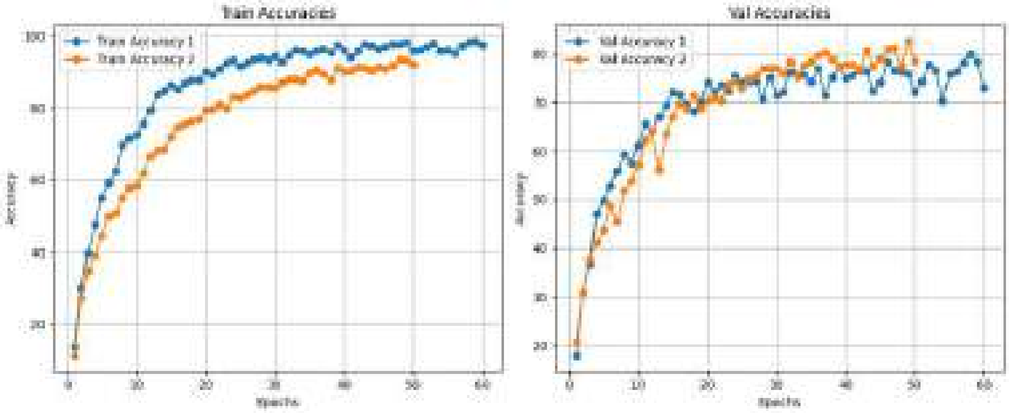
Orange = without ReLu

Blue = with ReLu

When comparing the model without ReLU (in orange) to the initial model, we noticed a slight drop in both accuracy and validation performance. This confirms the importance of ReLU in our network, as it introduces non-linearity and improves overall performance. Thus, we kept this layer in the model, as it enhances performance without significantly increasing computational complexity.

Dropout

One of the problems we faced during training was overfitting, which began around iteration 30. To mitigate this, we added a Dropout layer to the fully connected part of the model. Dropout works by randomly deactivating a proportion of neurons during training, which introduces noise into the data and helps prevent the model from overfitting to the training set. We tested a Dropout rate of 0.5.



Legend:

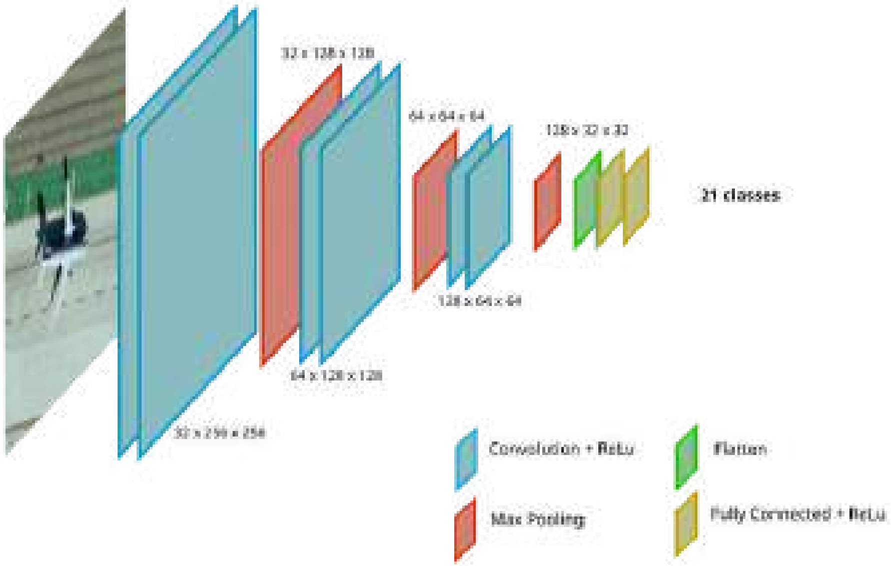
Orange = Dropout (0,5)

Blue = without Dropout

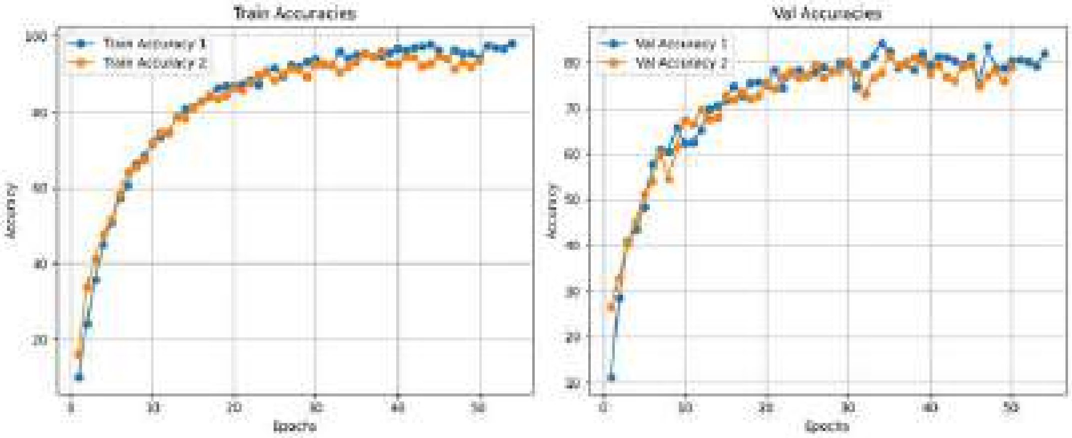
The results show that the model with Dropout no longer suffers from overfitting, and its validation performance improves over time. However, this model requires more iterations to achieve results comparable to the model without Dropout, which could be problematic for more complex models as training times increase.

Number of Convolutional Layers

In classical CNN architectures, it is common to use several convolutional layers before applying a max-pooling operation. We experimented with various models, including architectures with two consecutive convolutional layers before each max-pooling step as illustrated below.



This new model was then still trained on the same training and validation set.



Blue = Initial Model | Orange = Two Convolutions

After training this new model with two convolutional layers (in orange), we observed performance similar to the initial model in both accuracy and validation. However, this model required slightly more training time. Since we didn't find any adjustments that led to significant performance improvements, we decided to stick with a simpler model using just one convolutional layer per stage.

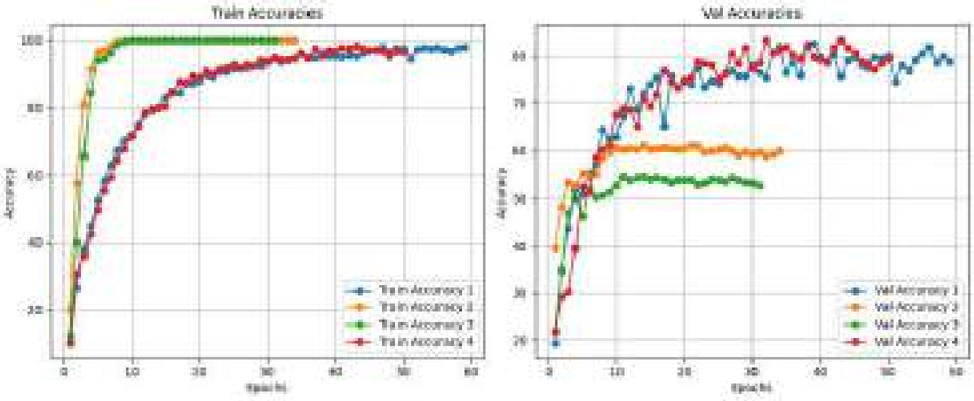
We will not elaborate further on the various parameters we tested, but it's worth noting that we also conducted many experiments regarding the fully connected layers to optimize the model.

Impact of the Sample

The test sample on which we train the data is crucial and should not be overlooked. This is why we conducted several tests on these data, comparing different optimization algorithms and various parameters such as the learning rate and the compromises to be made. In this section, we highlight the different transformations that can be applied to the data to improve the model's performance.

Transformations

To make the model more robust against the various images it may encounter, it is essential to broaden the range of images it can work with. Therefore, we applied several transformations to the data: a random horizontal flip, a random rotation, and normalization of input values. We test these different transformations applied to the initial model here. After multiple training sessions, we obtain the following results:



Blue: initial model = random flip, rotation and normalization

Orange: no transformations

Green: normalization only

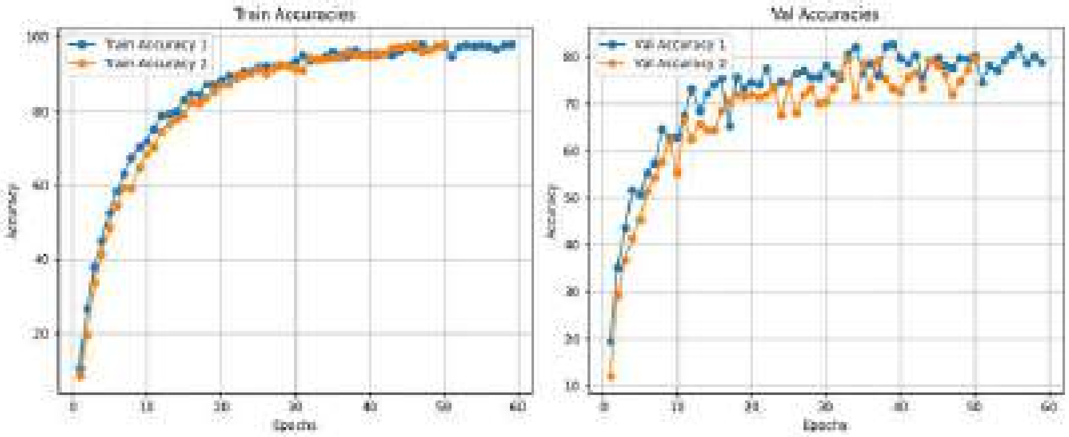
Red: random flip and rotation

We can observe two groups of results. In the case where no transformations are applied (in orange), the model converges very quickly after a few iterations, but the validation accuracy remains low at around 60%. Similarly, applying only finer normalization yields equivalent results, with validation around 55%. However, when we apply random rotations, the model achieves significantly better performance, reaching values comparable to those of the initial model.

It is important to note that these results are specific to the initial model, and for other models, the results may differ. Additionally, other transformations exist, such as random cropping, which could also contribute to improving the model's performance.

Data Separation

All available data is divided into three groups: training (70%), validation (10%), and testing (20%). However, the images are separated randomly based on their class. Given the limited number of images per class (approximately one hundred), it is possible that some classes may be underrepresented in certain groups. Therefore, we aimed to test whether equitably separating the data, ensuring that the same number of classes is represented in each dataset, could lead to any changes in performance.



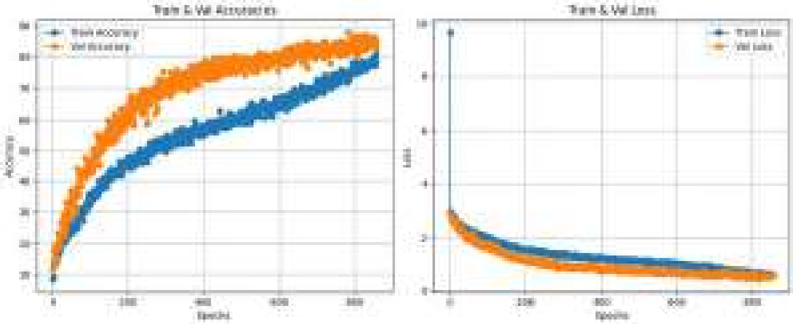
Blue: Full Random Separation | Orange: Data Separation

After observing the results of training and testing (not presented here), we noted performances very similar to those previously observed with the initial model. Notably, the classes that had difficulties with the initial model continued to show the same issues in this new model. We can assume that the size of the proposed sample was already sufficient to ensure a balanced distribution of data across the different datasets.

Optimized Model

Following all the observations we made, we identified several possible improvement paths for this model. We also understood that there is a trade-off between computation time and model performance. A model that converges quickly in a few iterations often yields mediocre validation performance, and consequently in testing. Conversely, models with dropout or Batch Normalization may take longer to adjust their parameters but can provide very interesting results over time.

Thus, we aimed to test the application of dropout and Batch Normalization on the initial model, along with other minor adjustments that we found relevant during training. This model requires a proper adjustment of the learning rate. Indeed, with a learning rate that is too high, the model did not progress and remained stuck at around 5% validation accuracy. On the other hand, a learning rate that is too low would significantly prolong the training time to achieve a conclusive result. Nevertheless, we decided to keep a low learning rate to observe the potential progress of this model.



```
----------------- Training Results -------------------
Final Training Accuracy: 78.37%
Final Validation Accuracy: 82.86%
Final Training Loss: 0.6006
Final Validation Loss: 0.5783

Best Validation Accuracy: 88.10% at epoch 782
```

We find here a rather surprising result, as the validation accuracy exceeds that of the training accuracy, even after 250 iterations, which is generally a good sign indicating that the model has not yet reached its full potential. Although it experienced a slowdown in its progress starting from iteration 200, the validation accuracy achieved a better level than that of the classical model and continues to progress slightly, reaching a peak of 88%.

In this section, we explored the challenges and strategies associated with single-label classification using convolutional neural networks (CNNs) on the UCM dataset. We implemented a basic CNN model and examined its performance through various adjustments, including the number of layers, channels, kernel sizes, and the application of techniques such as Batch Normalization and dropout. Our findings indicate that while a more complex model can enhance validation accuracy, careful tuning of hyperparameters and an understanding of the underlying data are crucial for optimal performance. Ultimately, our optimized model demonstrated promising results, achieving a validation accuracy of 88%, highlighting the potential for further improvements and refinements in future work.

03

RESNET

Context

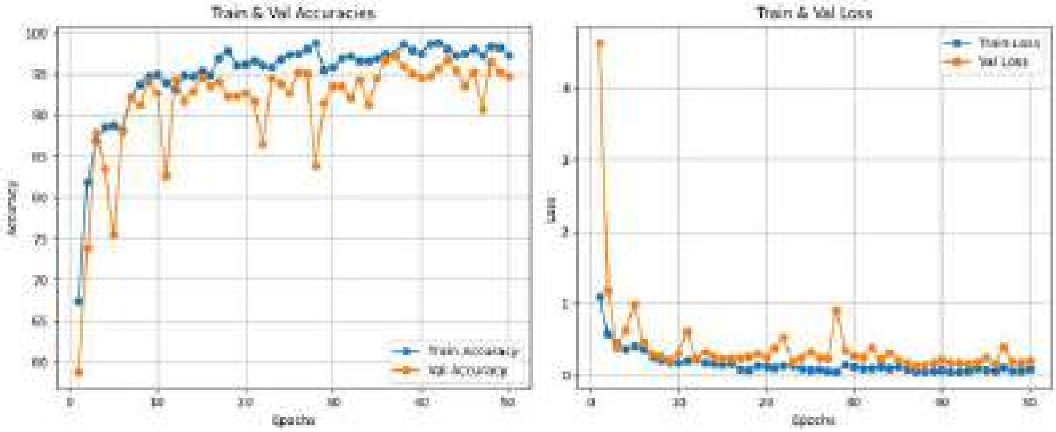
As we have observed, the model we proposed can still be improved. To achieve powerful models, a common practice called transfer learning is often utilized. As the name suggests, this method involves using pre-trained models and adapting them to new data, allowing us to leverage the knowledge gained during their previous training. In this section, we will use the ResNet model, which is widely recognized for its effectiveness in image recognition.

Configuration

With PyTorch, installing such a model is relatively straightforward. Once the model is imported, we just need to define how our fully connected layer will behave. This step is crucial, as we need to adapt it to a specific number of classes. In the case of single-label classification, there are 21 classes to consider.

Comparison

We trained the ResNet model on the same dataset as before and obtained the following results:



Right from the start, we notice that the model converges quickly, reaching a very acceptable value within about ten iterations. Similarly, the loss function decreases rapidly. Furthermore, the values obtained during training and validation are close, indicating that the model is not overly fitted.

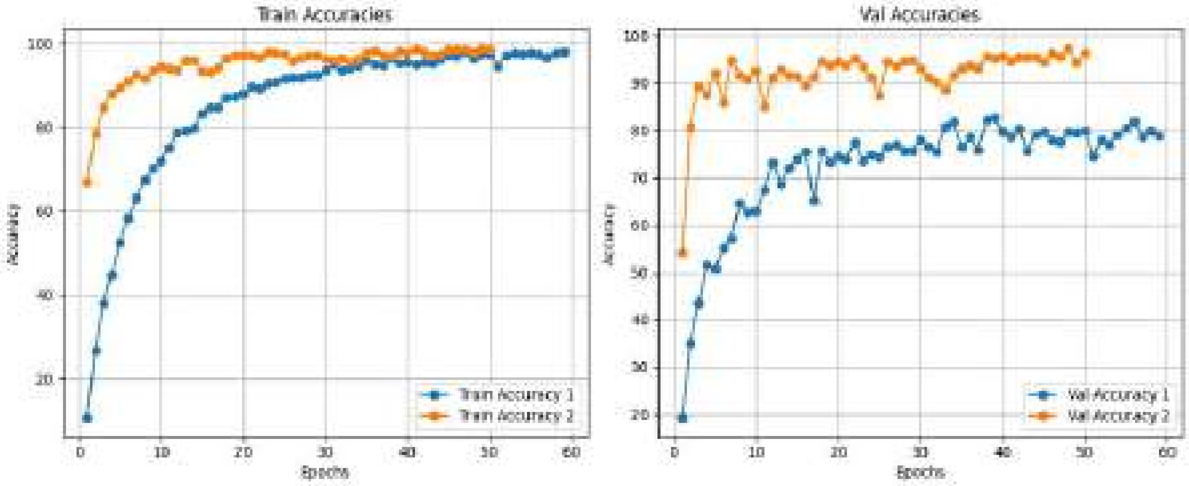
In detail, the results obtained are as follows:

---------------- Training Results ------------------
Final Training Accuracy: 97.21%
Final Validation Accuracy: 94.76%
Final Training Loss: 0.0889
Final Validation Loss: 0.1954

Best Validation Accuracy: 97.14% at epoch 37

With a maximum validation accuracy of 97%, the results of this training are very satisfactory.

When comparing this model with the baseline model, we can observe the following graphs:



Blue: Initial Model | Orange: Resnet

We again recognize the effectiveness of such a method, particularly in validation, where the results significantly surpass those of the classic model.

Moreover, with rapid convergence and satisfactory results, ResNet offers major advantages. To better compare the outcomes and to thoroughly examine any persistent issues and potential sources of error, we need to move on to the testing phase of the models.

04

RESULTS

Testing a Model

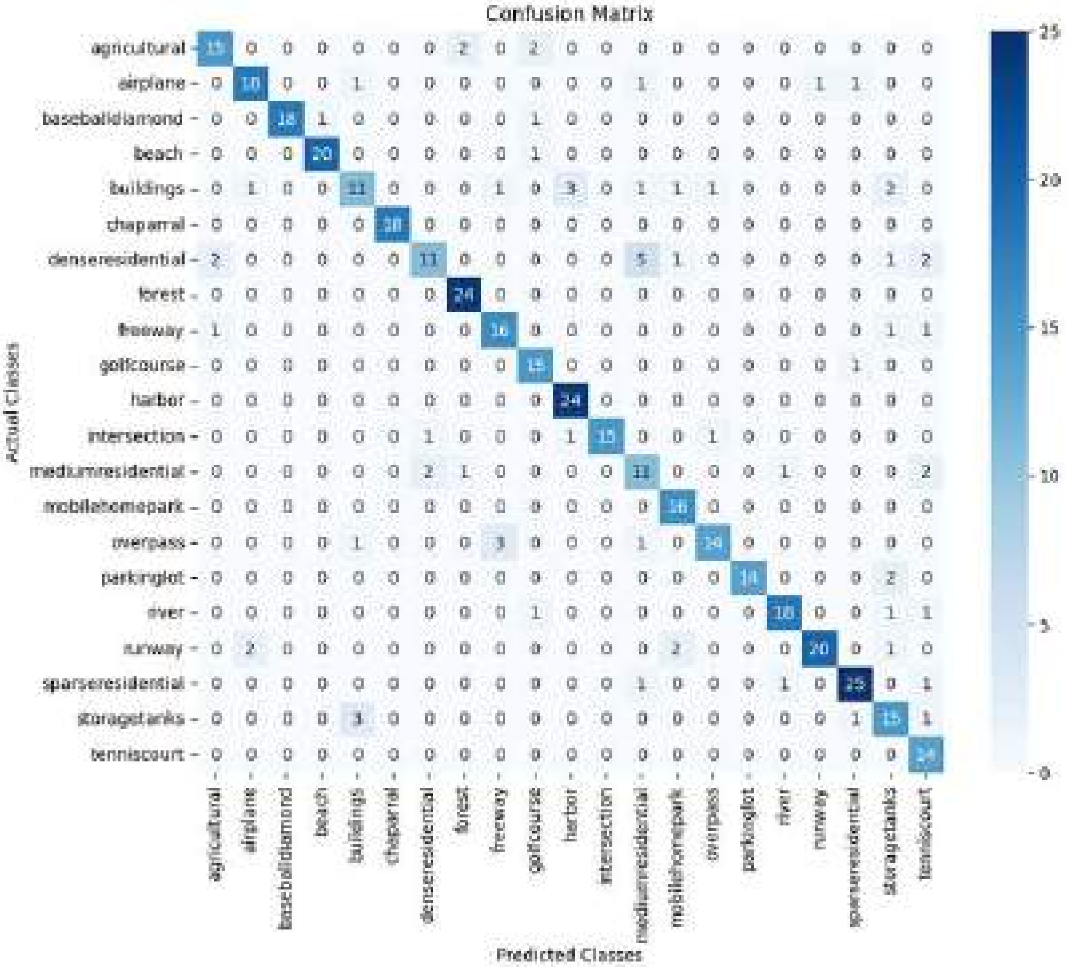
Validation serves as a good indicator during the training phase, as it assesses the real effectiveness of a model. However, to concretely verify results in real-world conditions, it is essential to conduct tests using the third dataset and analyze the strengths while addressing the weaknesses.

Single Label

Testing is performed similarly to validation, using the test dataset. Below, we present the results obtained for the initial single-label classification model.

---------------- Testing ------------------
Test Loss: 0.8255, Test Accuracy: 83.81%
Correctly classified samples: 352/420

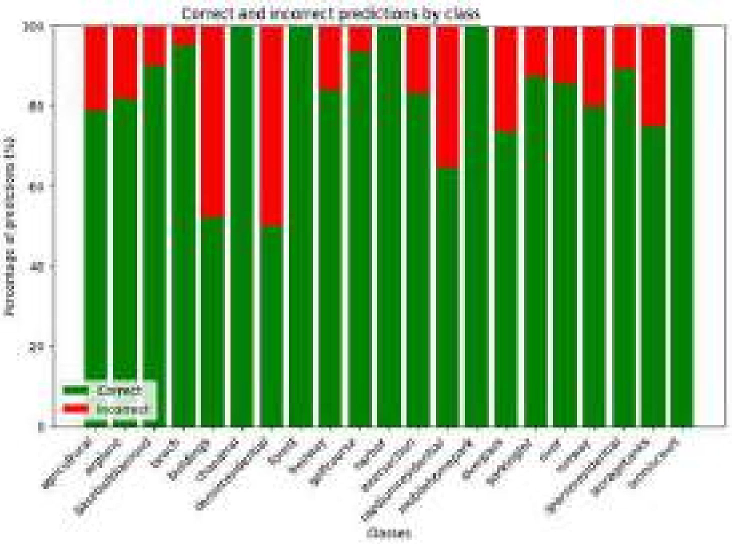
With an accuracy of nearly 84%, the test results are comparable to those achieved during the validation of this model. A good way to analyze the details of this performance is to compute the confusion matrix. This matrix will allow us to visualize classification errors, identify misclassified classes, and gain insights into areas for improvement.



It is observed that during this test, certain cases posed challenges, such as the denseresidential class, which is sometimes misclassified as medium residential.

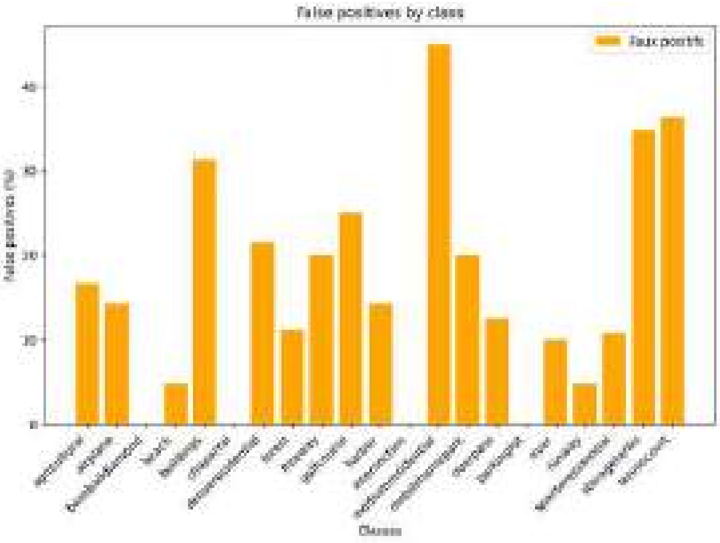
The following graphs provide a clearer visualization of false negatives and true positives.

For true positive, meaning when a class is correctly predicted, we have the following results:



It is immediately apparent that the building and denseresidential classes yield very poor results, with about a one-in-two chance of being correctly recognized. In contrast, classes like tennis court are always correctly identified.

On the other hand, false positives represent the reliability of the predictions. For example, although tennis courts are always correctly classified, there is a 33% chance that when the model predicts a tennis court, it may not actually be one.



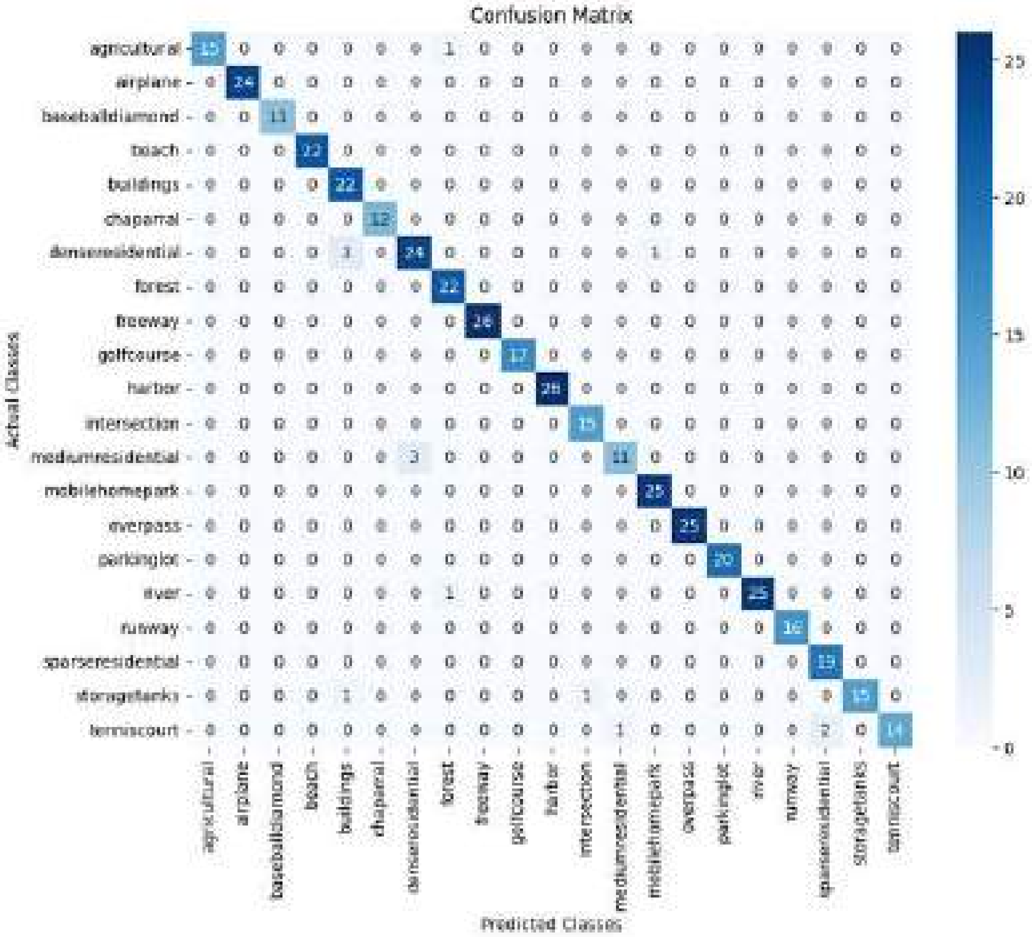
Depending on what is being studied, we may favor false negatives or false positives. This analysis can also provide insight into what additional training data might be necessary. It is worth noting that there is a slight bias, as not all classes have the same number of instances.

Resnet

We now repeat the tests on the ResNet model trained earlier. After testing, we find results similar to those obtained during validation.

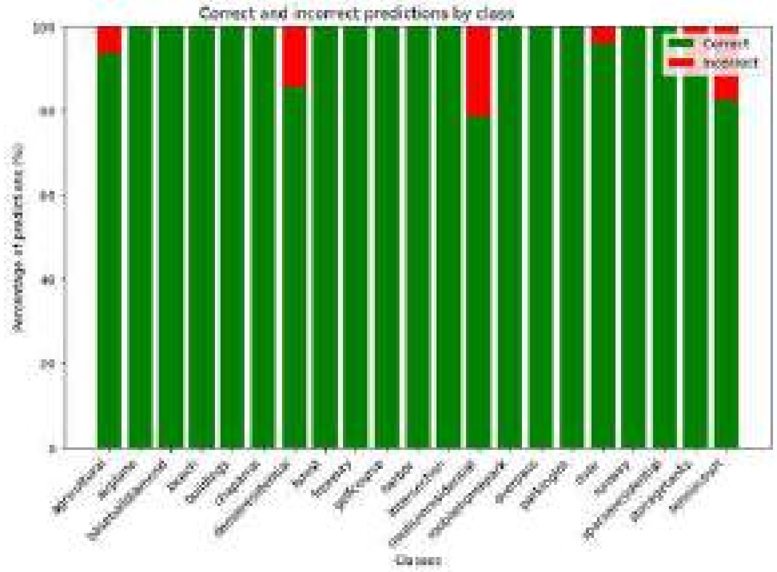
---------------- Testing ------------------
Test Loss: 0.1652, Test Accuracy: 96.67%
Correctly classified samples: 406/420

With an accuracy of 97% and only 14 misclassified images out of 420, the scores are once again very satisfactory. This is confirmed by the confusion matrix:

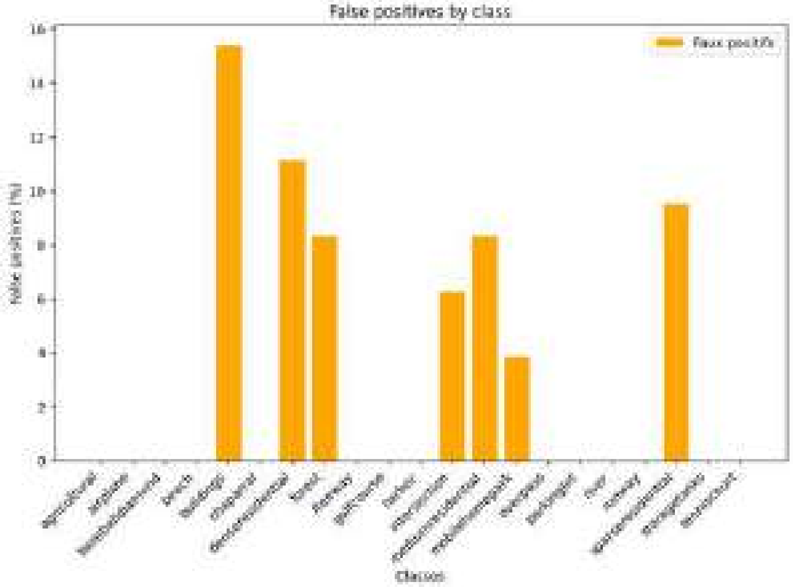


Apart from 3 denseresidential misclassified as buildings and a few mediumresidential confused with denseresidential, all other predictions are quite accurate, with the remaining errors being isolated cases.

When examining the distribution of true positives, this observation holds true.



Additionally, the false positives show much better results:



The worst case is for the buildings class, where 15% of predictions for this category are incorrect. However, many other classes, such as agriculture, airplanes, etc., have a reliability rate of 100%.

In conclusion, the ResNet model delivered excellent results with notable efficiency, unlike our base model. This clearly highlights the effectiveness of transfer learning, an approach that should certainly be considered for future CNN-based classifications.

05

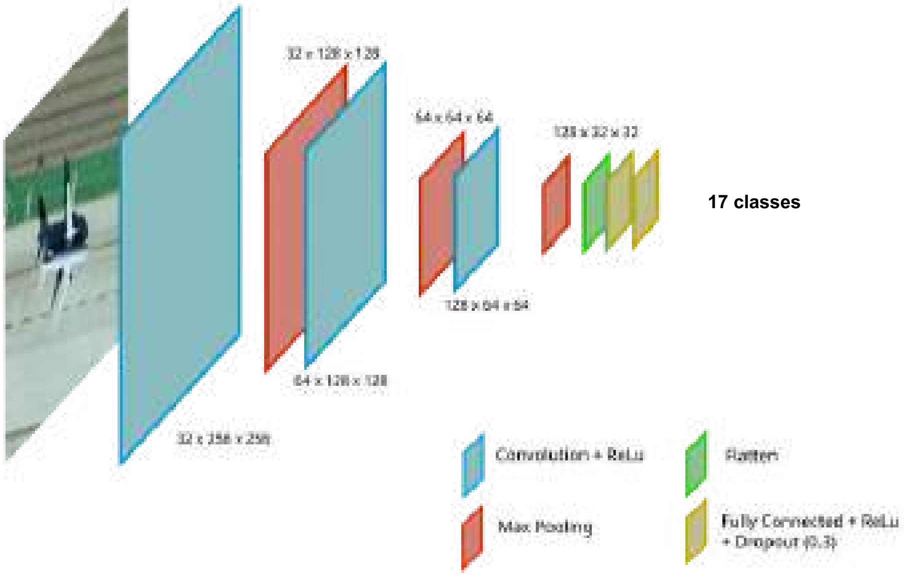
MULTI LABEL

Context

A second part of our project focused on multi-label classification, where each image can be associated with multiple categories. This dataset includes the same images as in the previous single-label task, but this time they are classified based on the presence of various objects and scenes. For example, labels such as "airplane," "trees," or "cars" represent objects, while labels like "sea," "grass," and "sand" denote landscapes. The challenge is different from single-label classification because instead of predicting the most likely single class, we must determine a threshold to decide whether each class is present or absent in the image.

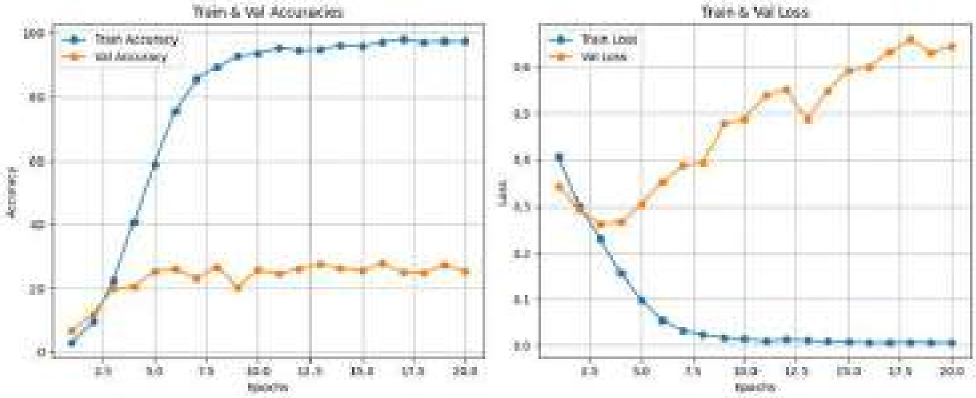
Model

We started by adapting the single-label model, modifying its fully connected layer to suit the multi-label task. Additionally, we incorporated a Dropout layer with a dropout rate of 0.3 to help prevent overfitting. For reference, the overall architecture of this adapted model is similar to that used for the single-label task but with this slight adjustment for multi-label classification.



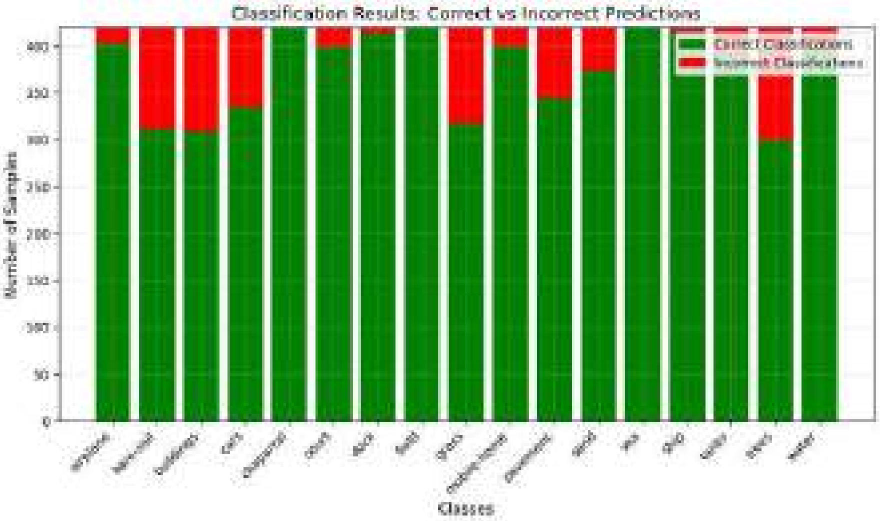
Results

We then trained this model on a new dataset specifically prepared for multi-label classification.



One of the first observations is the rapid convergence of the model after only a few iterations. However, validation accuracy appears low at first glance, hovering around 27%. We will revisit the reasons for this later. It's important to note that for an image to be considered correctly classified in the above graphs, all present classes must be identified—no more, no less. When you will have a look at the graphs below, keep in mind that we consider a correct true positive if the model managed to find the right class for a class that was represented in the picture, (even though he might not have gotten all the classes in the picture, which justifies his low score above). Additionally, the loss function for validation diverges, which is not a good indicator, likely signaling overfitting in the model.

The test results are as follows:

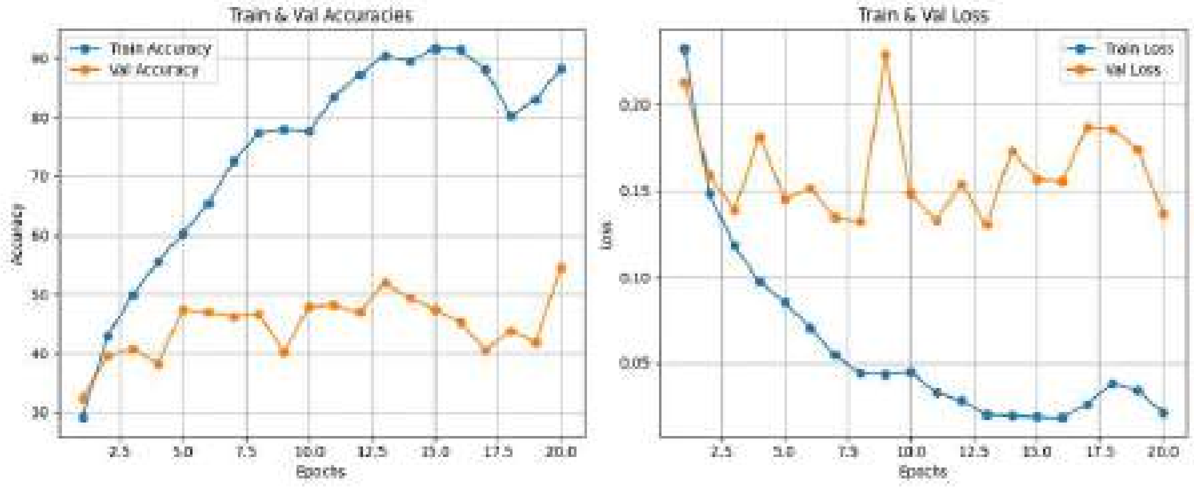


---------------- Testing ------------------
(These numbers consider that all classes were found)
Test Loss: 0.5996
Test Accuracy: 27.86%
Correctly classified samples: 117/420

The test confirms the validation results, with overall accuracy below 28%. However, certain classes, such as "sea," "fields," and "chaparral," are consistently well recognized. On the other hand, the presence of trees is correctly predicted only 75% of the time.

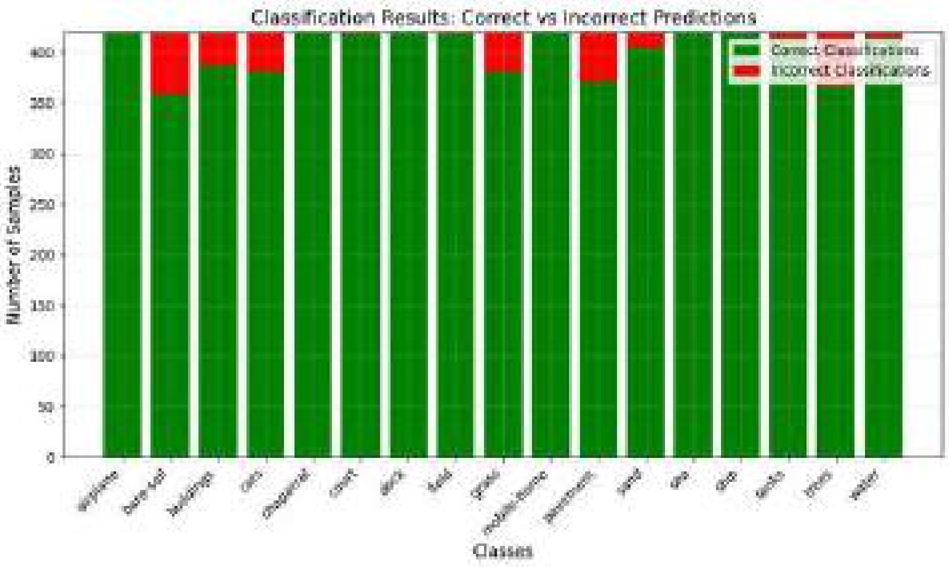
Resnet Multi

As we did with the single-label classification, we sought to enhance the multi-label model by applying a pre-trained ResNet model to the task. The results of training this model are as follows:



Although the model converged slower than in the single-label case, its performance remained solid, with accuracy not surpassing 90%. However, the validation score improved slightly with each iteration, reaching a significantly better score of 53% in the final epoch.

After running similar tests, we obtained the following results:



---------------- Testing ------------------
(These numbers consider that all classes were found)
Test Loss: 0.1371
Test Accuracy: 54.52%
Correctly classified samples: 229/420

When evaluating each category individually, the results are quite satisfying. Strong points include the detection of airplanes, docks, and boats, which were consistently recognized. However, a notable challenge was with "bare-soil," which was correctly identified only 85% of the time. Due to time constraints, we did not perform further tests, but we can still draw some conclusions from these initial results.

Why is the success rate lower than expected?

There are several explanations for this. First, multi-label classification is inherently more complex. The presence of multiple objects within an image introduces an extra layer of difficulty, as the model must determine not just the presence of one object but several. This adds a new parameter, increasing the likelihood of error.

Another reason is that, for an image to be considered correctly classified, all classes must be identified correctly. With 17 classes in total, if we assume that the precision for detecting any given class is around 96.5% (result from the first ResNet), the probability that all classes in an image are identified correctly is:

$$96.5\%^{17} \approx 54\%$$

Thus, an image has approximately a 54% chance of being classified perfectly. This is consistent with the test results we obtained. While individual class recognition rates are good, the probability of having at least one detection error per image is high. Multi-label classification therefore poses a significant challenge. In this example, there were only 17 possible classes, but if hundreds of classes were present, the potential error rate would increase significantly.

06

CONCLUSION

Conclusion

& acknowledgements

This project has allowed us to deeply understand the challenges and opportunities of using convolutional neural networks (CNNs) for the automatic classification of aerial images. Through the various stages of model design, training, and evaluation, we explored both single-label and multi-label classification. By employing approaches such as transfer learning with pre-trained models like ResNet, we achieved significant performance gains compared to models built from scratch.

Our results demonstrate that while CNN models are highly effective in aerial image recognition, challenges remain, especially in multi-label classification, where the simultaneous presence of multiple objects complicates the task. We also emphasized the importance of class balancing, data transformations, and parameter tuning, such as learning rate adjustments, to improve model robustness.

This work opens many avenues for further improvement, whether through the use of more sophisticated models or exploring alternative approaches like reinforcement learning or generative adversarial networks. In this sense, the project has been an excellent opportunity to learn and apply the concepts covered in the course while applying these techniques to real-world data.

Lastly, we would like to extend our thanks to Diego Marcos for his insightful lectures and for taking the time to read and evaluate our report. Though the lectures were few, they were of high quality and enabled us to develop strong skills in deep learning and image recognition.

DSA PROJECT