DEEP LEARNING PROJECT

Master in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

Group 12

Rita Palma 20240661,

Santiago Taylor 20240542,

Martim Tavares 20240508,

Maria Inês Assunção, 20211605

2nd Semester 2024-2025

Table of Contents

Introduction

Deep learning algorithms have gained popularity within the field of machine learning due to their performance and versatility, becoming a cornerstone for complex artificial intelligence. Among many of the tasks that these can perform, convolutional neural networks (CNNs) have provided advancements within the field of computer vision, allowing for predictions of different types of data, such as text, images, and audio. The automation of species identification, data collection and analysis, as highlighted in the study *Deep neural network identification of Limnonectes species* [2], shows how CNNs can be used in several fields, namely the field of biology.

To show how these can be useful for species identification, this project, developed for the Deep Learning course of Nova IMS, aims to explore the potential of these models at classifying rare species based on images sourced from the Encyclopedia of Life (EOL) and curated as part of the BioCLIP: A Vision Foundation Model for the Tree of Life [1]. This presents a multiclass image classification problem of over 200 rare species, for which different CNNs will be created using the Keras package for python, tested and tuned to predict the family of an animal given only an image. The effect of network architecture, image preprocessing and parameter tuning will be explored to see what impacts the predictive qualities of a CNN given the dataset, as well as how these perform against pre-trained models available online designed for similar tasks.

Data Exploration and Understanding

The initial dataset contains 11983 images of 202 distinct taxonomic families, of which all belong to the animalia kingdom and to 5 different phyla. *Image 1* shows a sample of 30 images from the dataset. As can be seen, the images come in many different formats (photos, drawings, fossils) and are taken from many different angles. The majority of classes were composed of 29-90 different images, with some classes having a higher frequency of up to 300 examples, represented in *Graph 1*. The overrepresentation of certain classes posed a risk, and this was kept in mind when designing models

Data Preparation

The database was split into sets for training, validation, and testing, consisting of 70% of the data, 15% and 15% respectively. The split was performed with stratification to maintain the same distribution of class. Image size would vary, so each image was downsized to a size of 224 x 224 pixels, as this was the standard seen for most pretrained models. Batch size was defined as 64 as this provided a balance between fast training and model performance, with images being shuffled beforehand. Lastly, the RGB values for each pixel were normalized to fit between the values of 0 and 1 for better processing.

Image Augmentation

To improve the performance and generalization capabilities of the model, several image augmentation techniques were explored, relying on the augmentation layers provided by the

package Keras. The different layers tested were: Random Brightness, Random Flip, Random Gaussian Blur, Random Rotation, Random Zoom, Random Cutout, Random Grayscale and Random Sharpness. These image augmentation techniques were chosen as they presented some of the possible real-world conditions that may affect photographs, such as lighting variations, different camera angles, perspectives and scales, and the presence of missing information. *Image 2* demonstrates the effect that different augmentations have on an image.

To test the effect of different augmentation combinations, a CNN (with two convolutional layers, two pooling layers and one dense layer) was trained with different augmentation pipelines. The training accuracy and validation accuracy were the chosen metrics to determine which of the 7 tested augmentation layer pipelines provided the best results. Additionally, the predefined Random Augmentation and Augmentation Mix layers were also tested (these include several augmentations in one layer) to further explore what performed better.

The two combinations that provided the best results were:

- Combination one: Brightness + Flip + Zoom + Cutout
- Combination two: Cutout + Grayscale + Flip + Blur

In addition to finding which augmentation layers performed best, Keras Tuner was used for the testing of hyperparameter combinations.

Model Development

The model development process, done through the Google Colab environment due to its performance, stemmed from an initial model, named *BasicCNN,* to which modifications were made. The sequential model's architecture was as follows: input layer, 2 convolutional layers, followed by batch normalization and 2 maxpooling, a flattening layer, a dropout layer with 0.5 and a dense layer.

The convolutional layers aim to capture abstract characteristic features that define each animal through a series of filters, with a ReLU activation function to add non-linearity to the layer outputs. The pooling layers reduce dimensionality and help with generalizing, improving the model's performance both in terms of accuracy as well as runtime. The dropout layer is used to increase the generalization of the model as well, before a dense layer used to predict the probability of belonging to a certain class (due to the "softmax" activation function). An Adam optimizer was used, and loss was calculated based on categorical cross entropy.

The next model *ParallelCNN* followed a similar structure, but implemented two parallel convolutional layers, to try and increase the accuracy of the model. The parallel layers are aimed at capturing diverse spatial patterns that a simple sequential model could not. Weight decay was used to regularize layer outputs, helping to generalize the model better.

Different callbacks were defined to support training:

- Early stopping: monitors validation accuracy and stops training if it doesn't improve for three epochs
- Learning rate scheduler: adjust the learning rate using an exponential decay function
- CSV logger: records training and validation metrics for later use

The model was trained with a sample of the train dataset (for faster run-time), and Combination 1 of augmentation was applied. It ran for 8 epochs after stopping due to validation accuracy not improving. The results for training and validation accuracy and loss, shown in *Image 3*, tell us that the model is underperforming and not generalizing well.

The model *ParallelCNN* was run a second time with oversampling of the minority classes (less than 90 images) to generalize better and address possible accuracy issues. After running for 15 epochs, this approach showed some improvements (*Image 4*), however the model still failed to learn general patterns. The validation F1 score is very low (0.0765), which indicates that the model is failing to make correct predictions for a large number of classes.

The model was run for a third time with the following alterations:

- Apply stronger augmentations: a third pipeline was created combining the original two
- Label smoothing (0.1) to reduce the model confidence in the predictions
- Add batch normalization after each convolutional layer, as a regularization technique
- Increase the dropout factor to 0.5

This one only ran for 6 epochs but it didn't show significant improvements.

Pre-trained Models

The ResNet50 is a popular deep learning CNN architecture for image classification. Its depth of convolutional layers allows for well-defined feature maps that improve prediction, at the cost of a longer run-time. For this reason, a sample of 80% of the training data was used for training. The layers from the pre-trained model were frozen, then a Global Average Pooling layer was implemented to reduce dimensionality, a Dense layer, a Dropout layer to reduce overfitting and a final Dense layer for classification. The model ran for 12 epochs before stopping early. It was then fine-tuned by unfreezing the last 10 layers to try and extract high level features. For this step, the learning rate was decreased.

The EfficientNetB3 is a pretrained model introduced by Google AI, with a focus on CNN efficiency. Due to this, the whole dataset was used for training. The layers from the pre-trained model were frozen, then a Global Average Pooling layer was added to reduce dimensionality, a Dropout layer of 0.4 to reduce overfitting and a Dense layer for prediction. Class weights were applied as a class

imbalance strategy. The optimizer used was Adam with a learning rate of 0.001 and it ran for 8 epochs.

Results

The results for each model is summarized in *Table 1*.

As can be seen, all models performed very poorly, whether they were pretrained or not:

- *BasicCNN* lived up to its name as being a starting point for other models but with very poor accuracy, essentially having no predictive capabilities.
- *ParallelCNN* performed well on the training data, with an accuracy of 37%, but failed to predict well on the validation data, hence a need to tackle overfitting
- *ParallelCNN_Improved* failed to improve upon the accuracy of the training data, and failed to improve the overfitting. Test accuracy was not measured due to technical issues, but validation data indicates that it would suffer from overfitting
- *ResNet50_Custom* performed with an accuracy of around 10% both on training as with the validation and test data
- *EfficientNetB3_Custom* provided the worst predictions from the pretrained models, predicting only 7% of validation data correctly. Due to time constraints, it was chosen not to test the test data

The best model was the *ResNet50_Custom* with an accuracy of 10%. This score is very poor, only correctly classifying 1 in every 10 images. Out of all the models, this model is likely to have performed best due to its many convolutional layers which are able to create feature maps capable of defining the images well in comparison with the other models tested. It is similarly the model that took the longest to train.

Conclusion

All models performed very poorly when considering the accuracy of the test data. The best model, being able to predict 1 in every 10 images, took the longest time to train and shows that the processing power of Google Colab was a big limiting factor in the potential of this project. The constraint on computational resources was a big challenge that was difficult to work around even with access to Google Colab due to their GPU use and run-time limitations. With *ResNet50_Custom* performing the best, it would be interesting to understand how the convolutional layers of the model work and what they identify.

Greater computational power would also allow for algorithms like Grid Search to help identify what hyperparameters would be best for each layer, likewise with the image augmentation, as this may

have also been a bottleneck for model accuracy and generalization. Class imbalance also posed an issue, and data augmentation could help with the generation of new images to improve sampling and training.

In its current form, the model presented would not be a great classifier of rare species, but the findings of how models work and what makes a model better provide solid ground for further exploration.

References

[1] Stevens, S., Wu, J., Thompson, M. J., Campolongo, E. G., Song, C. H., Carlyn, D. E., ... & Su, Y. (2024). Bioclip: A vision foundation model for the tree of life. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 19412-19424).
[2] Xu, L., Hong, Y., Smith, E. P., McLeod, D. S., Deng, X., & Freeman, L. J. (2023). Deep neural network identification of Limnonectes species and new class detection using image data. *arXiv*. https://doi.org/10.48550/arXiv.2311.08661

Annexes

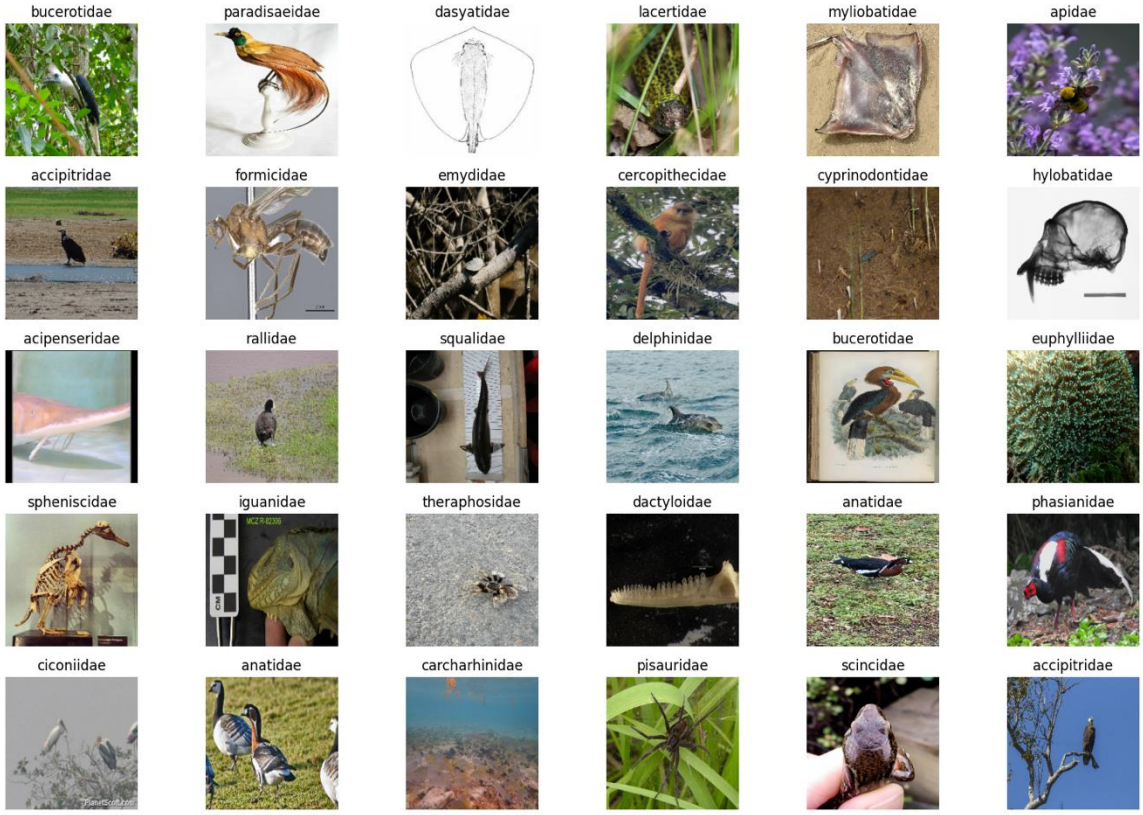
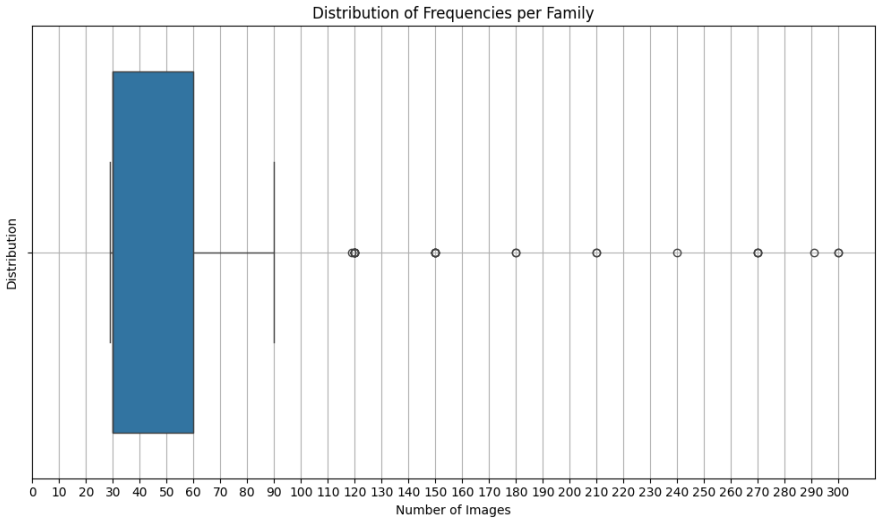


Image 1 – Examples of images from 30 different families.



Graph 1 – Box plot of distribution of family classes

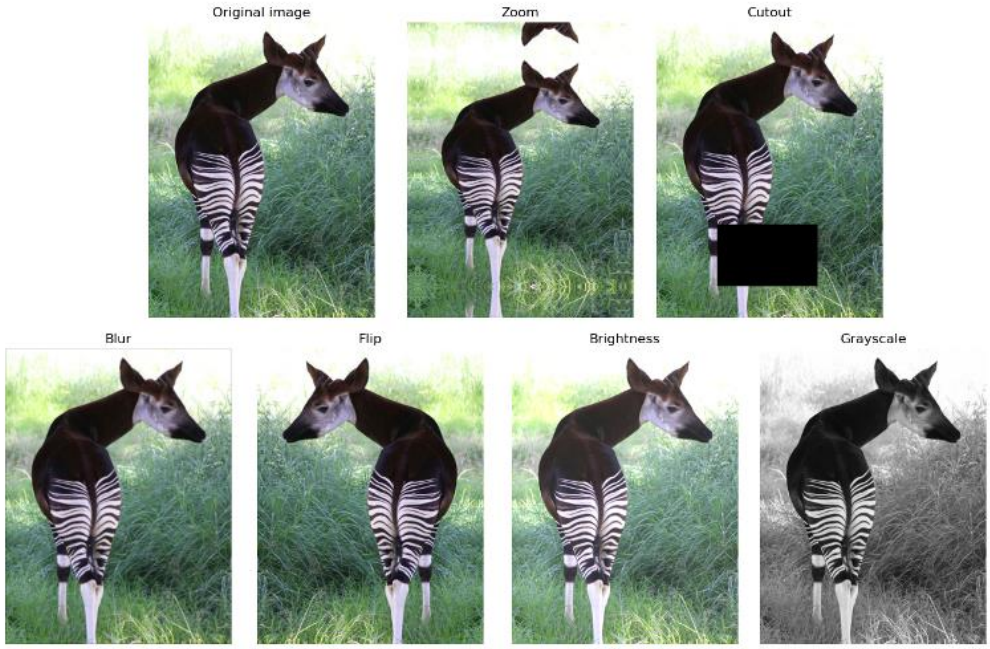


Image 2 - Effects of each augmentation

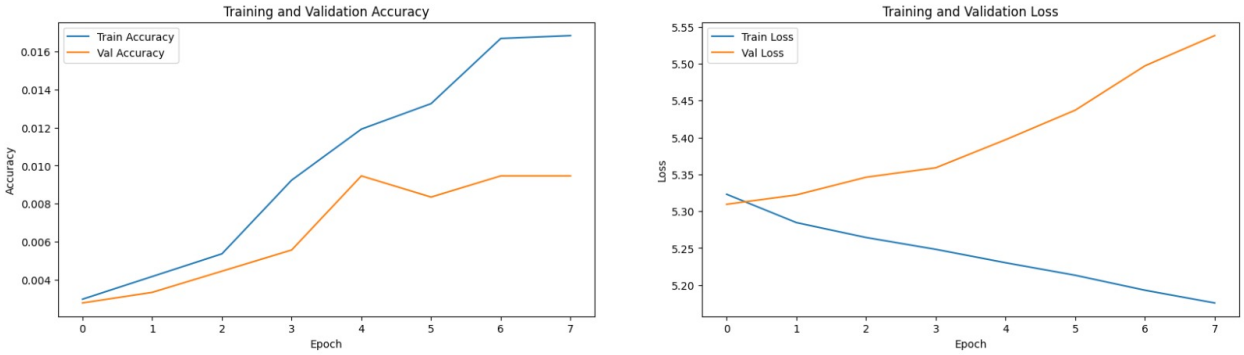


Image 3 - Training and validation accuracy and loss for ParallelCNN

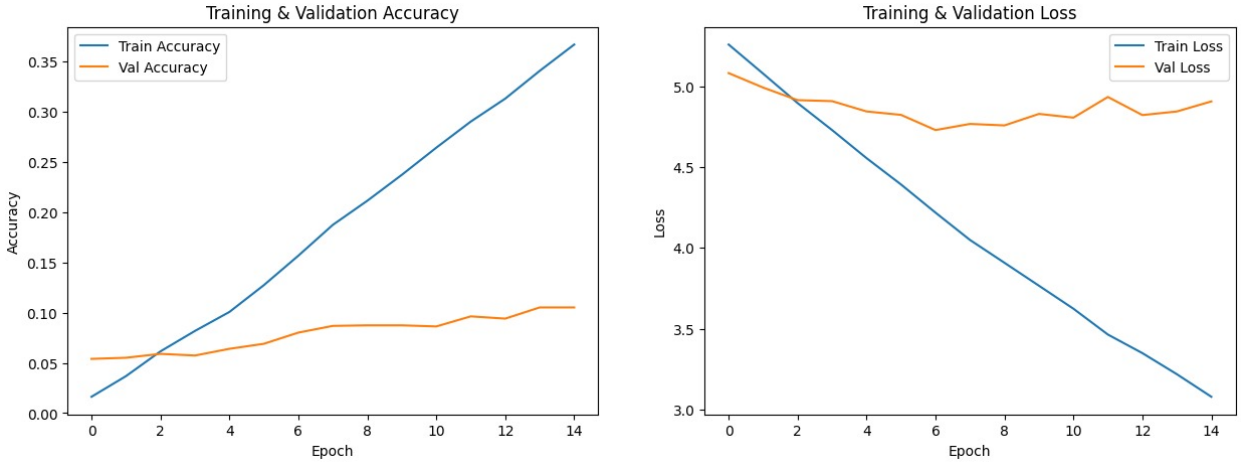


Image 4 - Training and validation accuracy and loss for *ParallelCNN* with oversampling.

Table 1 – Models Descriptions and Results

Model	Architecture	Image Augmentation	Accuracy	Loss
BasicCNN	- convolutional - batch normalization - max pooling - convolutional - batch normalization - max pooling - flatten - dropout (0.5) - dense	Brightness + Flip + Zoom + Cutout	Train_acc: ~0.01 Val_acc: ~0.02 Test_acc: ~0.02	Train Loss: 5.3 Val Loss: 7.8 Test Loss: 7.8
Parallel CNN (oversampled)	- convolutional - max pooling - 2 parallel convolutional - LeakyRelu activation - max pooling - flatten - dropout (0.3) - dense	Brightness + Flip + Zoom + Cutout	Train_acc: ~0.37 Val_acc: ~0.11 Test_acc: ~0.10	Train Loss: 3.1 Val Loss: 4.9 Test Loss: 4.8
Parallel CNN improved	- convolutional - batch normalization - max pooling - 2 parallel convolutional - LeakyRelu activation	Brightness + Flip + Zoom + Cutout + Grayscale + Blur	Train_acc: ~0.22 Val_acc: ~0.08	Train Loss: 4.2 Val Loss: 4.9

	- max pooling - flatten - dropout (0.5) - dense			
ResNet50_Custom	-ResNet50 pre-trained layers -Global Average Pooling -Dense, Dropout -Dense	Brightness + Flip + Zoom + Cutout	Train_acc: ~0.09 Val_acc: ~0.09 Test_acc: ~0.10	Train Loss: 4.5 Val Loss: 4.7 Test Loss: 4.6
EfficientNetB3_Custom	-EfficientNetB3 pre-trained layers -Global Average Pooling -Dropout (0.4) -Dense	Rotation + Width and height shift + Shear + Zoom, Flip + Fill	Train_acc: ~0.005 Val_acc: ~0.007 Test_acc: Not tested	Train Loss: 5.5 Val Loss: 5.3