Master's in Data Science and Advanced Analytics

NOVA Information Management School

Universidade Nova de Lisboa

Deep Learning Project

Report

Group 25

Bruna Simões, 20240491

Carolina Pinto, 20240494

Guilherme Cordeiro, 20240527

Marco Galão, 20201545

Margarida Cardoso, 20240493

Spring Semester 2024-2025

TABLE OF CONTENTS

1. Introduction 1
2. Data Exploration and Preprocessing 1
   2.1 Exploratory Data Analysis 1
   2.2 Preprocessing 1
      2.2.1 Oversampling 2
      2.2.2 Augmentations 2
3. Models From Scratch 2
5. Pre-trained Models 3
   5.1 Without *Phylum* Feature 3
   5.2 With *Phylum* Feature 5
5. Post-Model Optimization 5
   5.1 Outliers Treatment 5
   5.2 *Phylum*-based Taxonomy Constraint 6
6. Analysis of the results 6
7. Conclusion 6
8. References 7
Annex A: Visual Representations 8
Annex B: Tables 22
Annex C: Definitions 26
   1. MixUp 26
   2. VGGNet 26
   3. GoogleNet 26
   4. ResNet 27
   5. DenseNet 27

1. INTRODUCTION

This project focused on developing a deep learning model to classify the taxonomic family of rare species based on images sourced from the Encyclopedia of Life (EOL). It is a multi-class classification problem, having to classify 11,983 images along with metadata information, such as corresponding phylum, into 202 classes using a Convolutional Neural Network (CNN).

Throughout the project, various preprocessing, model architectures, and optimization experiments were developed and iterated upon to achieve our best model. This type of classification problem, with long-tailed distributions, is common in real-world biodiversity problems. Van Horn et al. (2018) in their study of the iNaturalist dataset highlights this challenge and emphasizes the importance of diverse techniques to improve model effectiveness [Bib. 1].

In this report, we will cover the steps taken in the analysis of the dataset, the approach for data preprocessing and transformation, the development of models from scratch, the implementation of transfer learning by leveraging pre-trained models, and model optimization efforts, as well as the rationale behind each decision and intermediary results. Finally, we will analyze the results achieved by our final model.

The full project is available on our GitHub repository [Bib. 2] for further exploration and collaboration.

2. DATA EXPLORATION AND PREPROCESSING

2.1 Exploratory Data Analysis

The dataset contains 11,983 images, each described by seven features related to the image source and species taxonomy. Since all samples belong to the *Animalia* kingdom, the most relevant features for classification are *phylum* and *family*, with *family* serving as the target label. To ensure robust model evaluation, the data was split into training (70%), validation (15%), and test (15%) sets.

There are 202 unique families, making this a multi-class classification problem. However, the dataset is highly imbalanced. As shown in the histogram and boxplot [Figure 1], a small number of families contain a large share of the images, while the majority are underrepresented. The mean number of images per family is 59, but the median is only 30, and the skewness is 2.72, indicating a strong right-skew. Such imbalance can cause the model to perform well on common classes but poorly on rare ones.

The dataset spans five phyla, with *Chordata* being the most common [Figure 2]. Since each family belongs exclusively to one phylum, we consider using *phylum* as an additional input feature in a multi-input model to reduce the effective number of candidate families per sample and improve classification accuracy.

In addition to class imbalance, we manual inspect random samples and identified anomalous images that do not depict any species. These likely originate from noisy or mislabeled data sources. To improve dataset quality, we will investigate image filtering strategies.

2.2 Preprocessing

To make the experimentation process more flexible and organized, we built a customizable preprocessing pipeline. This allowed us to try out different combinations of techniques – such as data augmentation and oversampling of underrepresented classes – and ensure that the essential preprocessing steps – like image normalization and target encoding – were consistently applied across all training runs, without having to rewrite code for each setup.

We normalized all images to ensure that pixel values were on the same scale, which improves convergence and helps the model train more efficiently. For the target variable, we used label encoding to convert the class names into integers, which is required for most classification models.

2.2.1 Oversampling

To address the pronounced class imbalance in the dataset we implemented an oversampling strategy at the batch level. This method aimed to ensure class representation across training batches while maintaining consistent batch sizes to avoid fluctuations in training dynamics and promoting stable gradient estimation – minimizing issues such as instability in the loss curve or ineffective gradient updates.

Minority classes were defined as those with fewer than 25 training instances, comprising 118 out of the 202 total classes. During each batch construction, we initially loaded a subset corresponding to approximately 75% of the target batch size rounded to the nearest integer (with a batch size of 32, we would have a subset of 24 images). Within this subset, instances from minority classes were duplicated on the fly to increase their representation without introducing a large memory overhead or requiring full dataset resampling.

Depending on the resulting batch size after duplication, one of two procedures was followed:

- **Excess case**: If the batch size exceeded the target size (e.g., 32 samples), the batch was first shuffled and then the most recently duplicated images were discarded.
- **Deficit case**: When the batch contained fewer samples than the target size, additional instances were added by randomly duplicating existing images from majority classes. If no majority class examples were available, duplication was performed using samples from minority classes.

Moreover, data augmentation - applied after the oversampling step and described in the next subsection - introduces stochastic variation into the duplicated samples. Since the applied augmentations involve random transformations, each duplicated image may undergo a distinct transformation, effectively simulating the presence of new samples. This approach provides many of the benefits of generating synthetic data, but without the computational cost typically associated with those techniques.

2.2.2 Augmentations

We implemented a wide range of transformations, grouped by intensity and type to help the model generalize better. These included geometric transformations (such as flips, rotations, zooms and translations), color adjustments (e.g., contrast, brightness, saturation, hue) and grayscale variations. We also integrated more advanced strategies such as *RandAugment*, which applies randomized combination of transformations, and *MixUp* (detailed in Annex C: Definitions), which blend pairs of images and their labels to generate smoother decision boundaries. A detailed summary of all augmentations is provided in Table 1.

3. MODELS FROM SCRATCH

This section focuses on exploring models trained from scratch, with randomly initialized weights, to evaluate their ability to learn directly from the dataset.

We initially tested five different architectures, all trained under the same conditions - using oversampling and without data augmentation. As a baseline we implemented a simple CNN consisting of two convolutional layers, max pooling, a dropout layer, and a fully connected softmax output layer. The remaining four models – GoogLeNet [Bib. 3], VGGNet [Bib. 4], ResNet [Bib. 5], and DenseNet [Bib. 6] - were adapted from architectures introduced in the course materials. Further architectural details are provided in Annex C: Definitions and their performance results are presented in Table 2.

Both the baseline model and VGGNet achieved a very high training f1-score (0.9974 and 0.8509, respectively). However, their inability to generalize is reflected in the low validation performances (f1-score of 0.1326 and 0.0978) - highlighting significant overfitting. Consequently, we excluded them from further analysis.

Given the results, to address the overfitting shown in the selected architectures, we introduced data augmentation into the training pipeline and evaluated its impact. As shown in Table 3, *MixUp* consistently

led to a more favorable trade-off between validation performance and generalization and was therefore adopted as the default augmentation strategy.

Although the models were based on established architectures, we implemented them from scratch and used them to explore structural modifications. This helped us obtain a better understanding of their behavior and define a more focused Hyperband tuning space. To further support efficient training and avoid overfitting, we employed two callbacks: *ReduceLROnPlateau*, which reduces the learning rate when validation loss plateaus, and *EarlyStopping*, which stops training and restores the best weights.

To better adapt the models to our classification task we simplified the original designs by reducing depth and, in the case of GoogLeNet, by removing auxiliary classifiers. ResNet and DenseNet were retained in forms close to their standard formulations, as both have been shown to generalize well in tasks with limited training data and large label spaces, owing to their robust gradient flow and efficient parameter usage. In contrast, GoogLeNet lacks such architectural mechanisms, making it less effective in these settings. To address this, we developed a modified version, introducing batch normalization, L2 regularization, and global average pooling. These changes reduced overfitting, reflected by a 10 percentage points decrease in the gap between training and validation f1-scores, while maintaining comparable validation performance (0.1359 vs. 0.1095) [Table 4]. For ResNet and DenseNet, we explored different depths and structural configurations, such as wide versus standard models and the use of residual versus bottleneck blocks in ResNet [Table 5], and variations in block depth for DenseNet, including a deeper DenseNet 151 and a version with fewer layers per block [Table 6].

The models selected for Hyperband tuning were those that demonstrated the best trade-off between validation performance and overfitting. Specifically, we chose the modified GoogLeNet, ResNet 18 (standard model with residual blocks), and the most simplified version of DenseNet. For instance, the modified GoogLeNet achieved a training f1-score of 0.2212 and validation f1-score of 0.1095, ResNet 18 achieved 0.1209 and 0.0671, and Simplified DenseNet achieved 0.1402 and 0.0825. These architectures provided the most promising starting points for tuning, balancing learning capacity with generalization. The full range of hyperparameter values explored during tuning is presented in Table 7 to Table 9, with corresponding results summarized in Table 10 to Table 12.

Overall, performance improvements were limited. The best-performing model - GoogLeNet after tuning (Hyperband 1) - achieved a training f1-score of 0.1971 and a validation f1-score of 0.1023, representing the most favorable trade-off between predictive performance and overfitting. DenseNet Simplified obtained a slightly lower validation f1-score (0.0825) but demonstrated the smallest gap between training and validation scores, indicating stronger generalization. ResNet18 exhibited minimal overfitting but the lowest validation f1-score (0.0379), suggesting limited capacity to distinguish between classes, and the results after tuning didn't demonstrate more improvements. These findings indicate that training from scratch in the presence of severe class imbalance and limited data diversity - even with oversampling - is insufficient for effective learning. In particular, the uniformly low macro f1-scores across models highlight persistent difficulties in learning to predict underrepresented classes, which constitute the majority of the label space. These findings emphasize the necessity of incorporating transfer learning to improve performance.

For additional visualizations illustrating the evolution of the explore models, refer Figure 3 to Figure 10.

5. PRE-TRAINED MODELS

5.1 Without *Phylum* Feature

We decided to implement three pre-trained models: ResNet50, VGG16, and EfficientNet, utilizing the TensorFlow and Keras libraries. The pipeline was adapted to use the Keras preprocessing functions

corresponding to each model, ensuring that images were scaled according to the models' original training setup, avoiding incompatibilities that could compromise performance.

After this, several architectures were tested and tried to put on top of the pre-trained model. The final head architecture was inspired by a Tensor Flow documentation example [Bib. 7] and it consists of a GlobalAveragePooling layer, followed by two Dense layers and a Dropout layer for regularization. This architecture provided a good trade-off between expressiveness and simplicity, while including a moderate number of trainable parameters, which is suitable given the small number of samples per class in the dataset.

This architecture was then modified to include regularization parameters, since during initial trials, and has exemplified by baseline model 1 (baseline model without regularization and without additional pre-processing), the models were exhibiting signs of overfitting starting as early as epoch 4. The Dropout Layer was included [Bib. 8], followed by L2 regularization to the dense layers of the models to prevent them from memorizing the training data. The choice of 1e-4 follows deep learning literature, as those outlined in Goodfellow et al. (2016), where small L2 penalties are shown to effectively regularize models without significantly impairing their learning capacity [Bib. 9]. Following Müller et al. (2019) [Bib. 10], we applied label smoothing with a value of 0.01 to prevent the model from becoming overconfident in its predictions, which encourages better generalization and improves the model's robustness to uncertain labels. The authors explored smoothing values in the range of 0.05 to 0.2, but we opted for a more conservative setting to avoid flattening the label distribution. This is important in our case, since the classification task involves 202 fine-grained and sparse classes. A lower smoothing value allows us to retain high fidelity in class distinctions while still benefiting from the regularization effect, which is crucial for maintaining the model's sensitivity to subtle inter-class differences.

Another overfitting countermeasure was to use the exponentially decaying learning rate function, so that the model made larger parameter updates in early stages, encouraging faster convergence, while smaller updates in later stages allow it to fine-tune weights, preventing overfitting. This approach has been shown to promote greater stability and generalization [Bib. 11].

With the architecture defined, we proceeded to train each of the three base models. We conducted exploratory training over several epochs to identify the one at which overfitting became apparent. Once this threshold was observed, the models were re-trained with adjusted stopping points to capture the best generalization performance. This way, we aim to capture each model's optimal generalization performance, ensuring a consistent basis for comparison.

After 25 and 50 epochs, respectively, both the ResNet and VGG16 models started to overfit. EfficientNet, when trained for 50 epochs, began to stagnate in all validation metrics, which were no longer keeping up with the evolution of the training metrics. Because of this, we decided not to train it further and proceeded to the fine-tuning stage with our best performing model – EfficientNet.

The fine-tuning stage consists of unfreezing previously untouched layers of the pre-trained model, in hopes that performance metrics improve. In the initial trials, however, the models started overfitting as early as the second epoch, meaning the unfreezing wasn't working as intended. In our view, this happened because unfreezing too many layers at once doesn't suit our problem — the dataset is small, very specific, and quite different from the data these models were originally trained on. To address this, we followed an approach inspired by Rahman et al. (2020), who recommend avoiding full unfreezing when working with small and domain-specific datasets [Figure 11]. Based on that, we chose not to unfreeze the entire model, and instead unfroze one layer every two epochs, to reduce overfitting while allowing the model to adapt [Bib. 12].

The second baseline model with regularization but without preprocessing was also compared to our final model. Although accuracy is similar, there is an almost 8% increase in Validation F1-score by using the custom mixup and oversampling strategy previously defined. [Table 13]

Analyzing the metrics plots [Figure 12] we can assure that the EfficientNet Fine-Tuned model provides a better balance between training and validation loss, f1-score and accuracy suggesting improved generalization and performance overall. Even it's not fine-tuned version, EfficientNet exhibits a better trade-off in all metrics when compared to both VGG16 and ResNet50. The structure of this model is presented in Figure 13.

5.2 With *Phylum* Feature

Given that each family in the dataset belongs exclusively to one of five phyla, we explored whether incorporating phylum information could support the classification task. The intuition was that adding phylum context might help the model narrow down the candidate families for each sample, effectively reducing class ambiguity and potentially improving performance.

To implement this idea, we introduced *phylum* as an additional input feature in our model pipeline. The first step involved encoding the categorical *phylum* variable using a *label encoder*, followed by one-hot encoding to obtain a five-dimensional binary vector representing each sample's phylum. These vectors were stored alongside the already one-hot encoded *family* labels.

The data preprocessing pipeline was extended to accommodate this new input. Specifically, the image loader and augmentation steps were adapted to process both image data and phylum vectors in parallel. This required adjusting the oversampling logic to ensure that both the image data and their corresponding phylum labels were padded or truncated together during batch construction. The *MixUp* augmentation strategy was also adjusted to combine the phylum vectors proportionally, reflecting the mixture ratio of the combined input images..

In the model architecture, a new input layer was added to receive the one-hot phylum vector. After extracting image features using a frozen EfficientNetB0 backbone and applying global average pooling, we concatenated the resulting image embedding with the phylum vector. Following the same approach as in the baseline model, this combined feature was then passed through a dense layer with *ReLU* activation and dropout regularization before reaching the final *softmax* output layer predicting one of 202 family classes.

We maintained consistency with the baseline model by keeping the training pipeline components unchanged, including data augmentation, optimizer settings, learning rate schedules, early stopping, label smoothing, and regularization. This ensures that any performance difference can be attributed solely to the architectural changes in the classification head.

Although this phylum-informed architecture was intended to enhance the classification process, its impact on performance turned out to be limited, revealing a worse performance [Figure 14 and Table 14].

5. Post-Model Optimization

5.1 Outliers Treatment

To improve model performance after initial training, we focused on filtering out potentially noisy images identified during the exploratory data analysis (EDA) phase. The optimization strategy involved defining a confidence metric for each prediction, selecting a threshold, and retraining the model on a filtered dataset.

We defined confidence as the predicted class's probability. Images with a confidence score below 0.15 were considered uncertain and filtered out. The threshold of 0.15 was chosen to be slightly conservative, avoiding the removal of too many images [Figure 15]. We applied this filtering in two trials: in the first, we removed only low-confidence misclassified images (Figure 16); in the second, we applied the filter to both correctly and incorrectly classified samples [Figure 17].

The first approach removed 264 images (3.15% of the dataset), while the second removed 405 images (4.83%). After retraining the model with both filtered datasets, results showed that the best performance was achieved with trial 1, as show in Figure 18.

5.2 *Phylum*-based Taxonomy Constraint

To enhance model performance, we implemented a prediction refinement step grounded in biological taxonomy. As each family belongs exclusively to a single phylum, we leveraged this hierarchical relationship to refine predictions. Specifically, rather than selecting the class with the highest softmax probability across all families, we constrained the model to choose the most probable family within the correct phylum of the true label. This biologically informed filtering step helps eliminate implausible predictions and enforces taxonomic consistency. As a result, model performance improved, with the validation F1-score increasing from 0.677 to 0.688 and accuracy from 0.701 to 0.710 [Figure 19].

6. ANALYSIS OF THE RESULTS

The confusion matrix [Figure 20] has mostly darker squares in the diagonal, which means that most samples have the prediction corresponding to their true label. Figure 21 and Figure 22 show that, for the most part, majority classes have a high percentage of correctly classified samples, highlighting the imbalance problem of our dataset. However, there are some minority classes that have an accuracy of 100%, suggesting that some rare species may have distinct visual features easily learned by the model.

We also carried out an analysis regarding misclassified samples and the confidence of our model. There are images with high confidence predictions that are incorrect. For example, the family Cracidae is predicted as Bucerotidae with a confidence of 0.91. Those are both bird species that have similar visual characteristics. This also happens with fish species, like Myliobatidae that is predicted as Dasyatidae with a confidence of 0.88. The similarity of these species is highlighted in Figure 23, Figure 25 and Figure 26. Species like Apidae and Formicidae are correctly predicted with a confidence of 1 [Figure 24 and Figure 27], probably because they exhibit highly distinguishable patterns.

7. CONCLUSION

This project set out to classify 11 983 images of rare species into 202 taxonomic families, addressing a severe class imbalance and a limited number of samples per class. The exploration of different preprocesing techniques – custom batch-level oversampling and augmentation, particularly MixUp - was essential to accommodate underrepresented classes without inflating memory requirements.

Training models from scratch was challenging, especially regarding overfitting and slow performance improvement. Our exploration focused on refining well-defined architectures, wich helped us to better understand the diversity and behaviour of Convolutional Neural Networks. Transfer learning with EfficientNet augmented by L2 regularization, label smoothing and an exponentially decaying learning rate, consistently outperformed both VGG16 and ResNet50 baselines. The fine-tunning by unfreezing one layer every two epochs was important to reduce overfitting and improve performance.

Post-training outlier treatment, removing low-confidence predictions (≤ 0.15) and enforcing a phylum-based constraint enhanced taxonomic consistency, increasing validation macro F1 from 0.677 to 0.688 and accuracy from 0.701 to 0.710. These post-model optimization efforts allowed us to get a deeper understanding of the dataset and enhanced model effectiveness.

Despite this efforts, minority classes remained challenging, since they still suffer from lower recall. To improve the results it would be interesting to explore hierarchical classification techniques, synthetic data generation for scarce taxa, and ensemble strategies to further mitigate class imbalance and improve model robustness.

8. REFERENCES

[Bib. 1] Horn, Grant, et al. *The INaturalist Species Classification and Detection Dataset*.

[Bib. 2] mgalao. “GitHub - Mgalao/Deep-Learning-Project.” GitHub, 2025, github.com/mgalao/deep-learning-project.

[Bib. 3] Szegedy, Christian, et al. Going Deeper with Convolutions. 17 Sept. 2014.

[Bib. 4] Simonyan, Karen, and Andrew Zisserman. VERY DEEP CONVOLUTIONAL NETWORKS for LARGE-SCALE IMAGE RECOGNITION. 10 Apr. 2015.

[Bib. 5] He, Kaiming, et al. Deep Residual Learning for Image Recognition. 10 Dec. 2015.

[Bib. 6] Huang, Gao, et al. Densely Connected Convolutional Networks. 28 Jan. 2018.

[Bib. 7] TensorFlow. (n.d.). Transfer learning and fine-tuning. Retrieved from https://www.tensorflow.org/tutorials/images/transfer_learning?hl=en

[Bib. 8] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.

[Bib. 9] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[Bib. 10] Müller, R., Kornblith, S., & Hinton, G. (2019). When Does Label Smoothing Help? In Advances in Neural Information Processing Systems (Vol. 32)

[Bib. 11] Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In Neural networks: Tricks of the trade (pp. 437–478). Springer.

[Bib. 12] Rahman, M. M., Sarker, I. H., & Nowrin, F. (2020). Transfer learning with deep CNNs for disease detection in plant leaf images. International Journal of Innovative Science and Research Technology, 5(11), 1227–1233.

[Bib. 13] Team, Keras. “Keras Documentation: MixUp Augmentation for Image Classification.” Keras.io, keras.io/examples/vision/mixup/.

ANNEX A: VISUAL REPRESENTATIONS

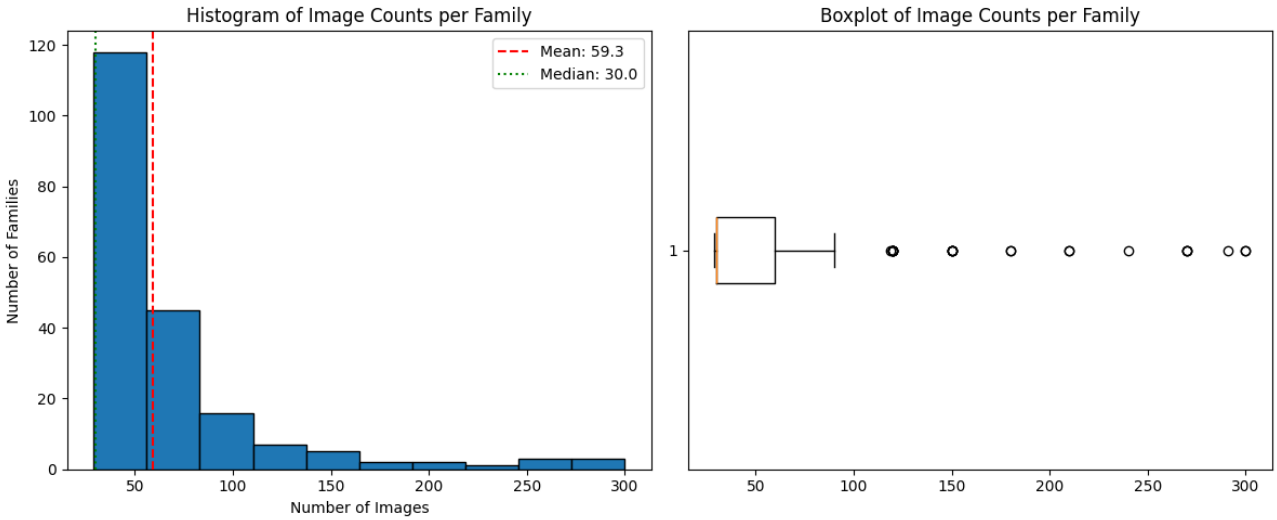


Figure 1 - Histogram and Boxplot of Image Counts per Family

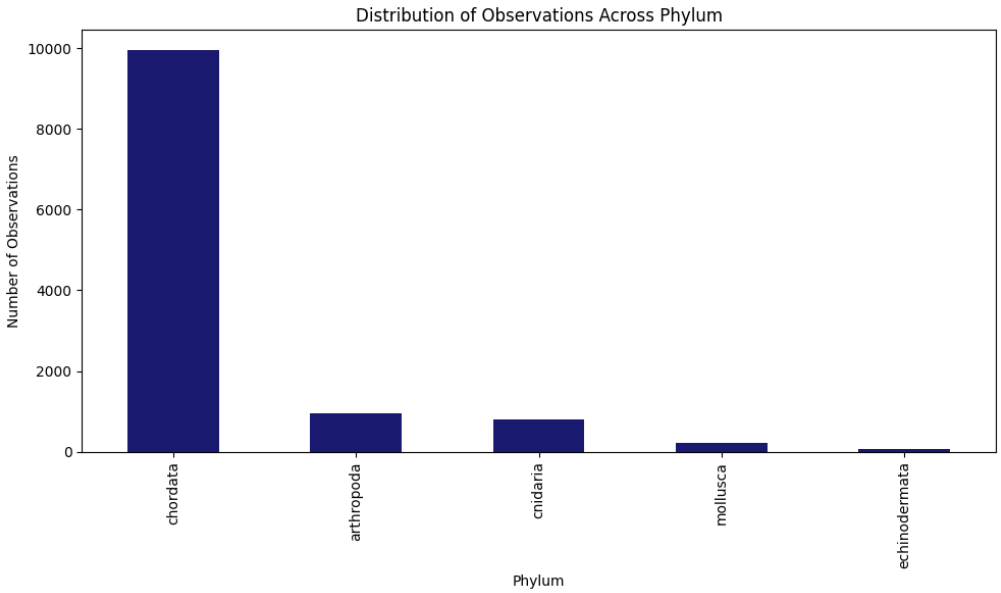


Figure 2 - Distribution of Samples Across Phylum

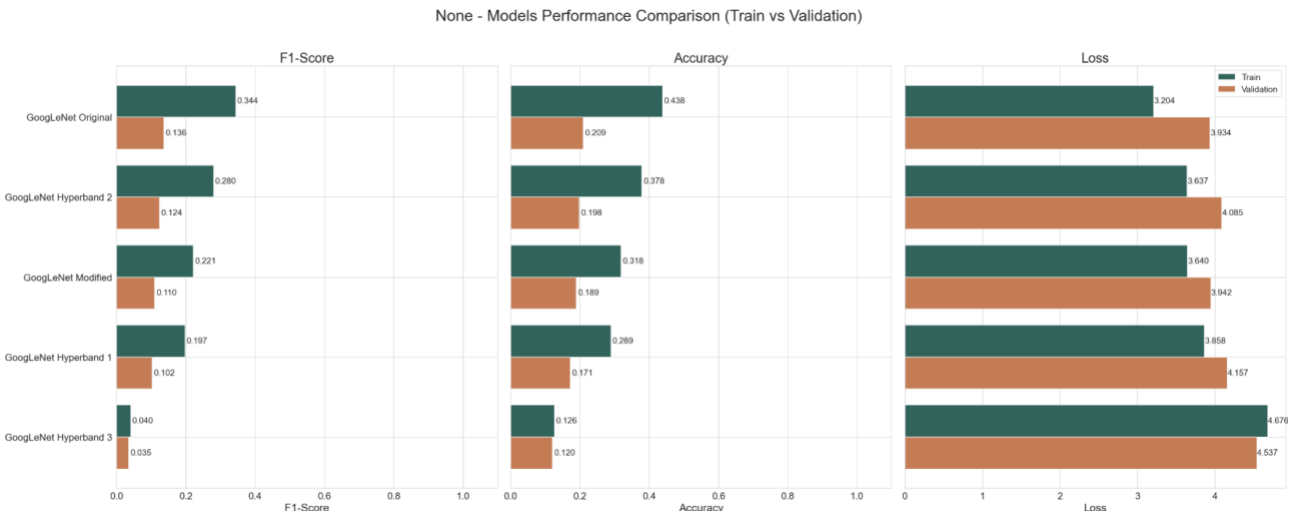


Figure 3 - Visual comparison of the performance metrics across GoogLeNet models

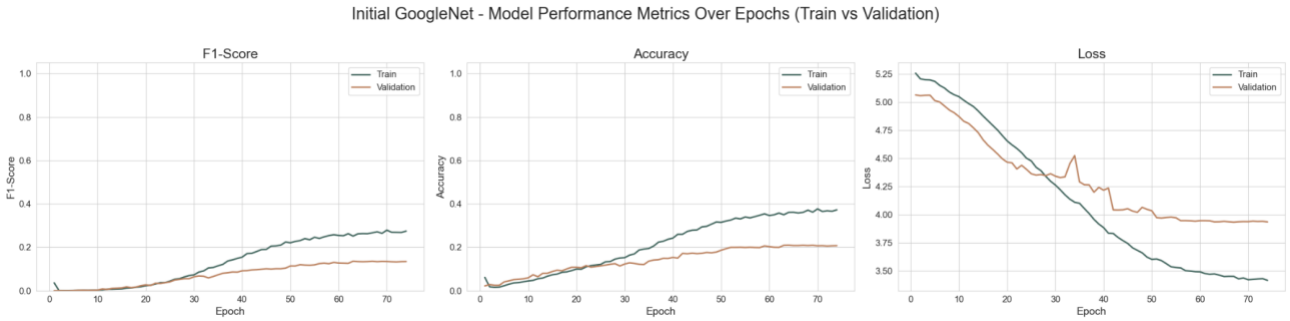


Figure 4 - Training and validation performance of the original GoogLeNet model over 75 epochs

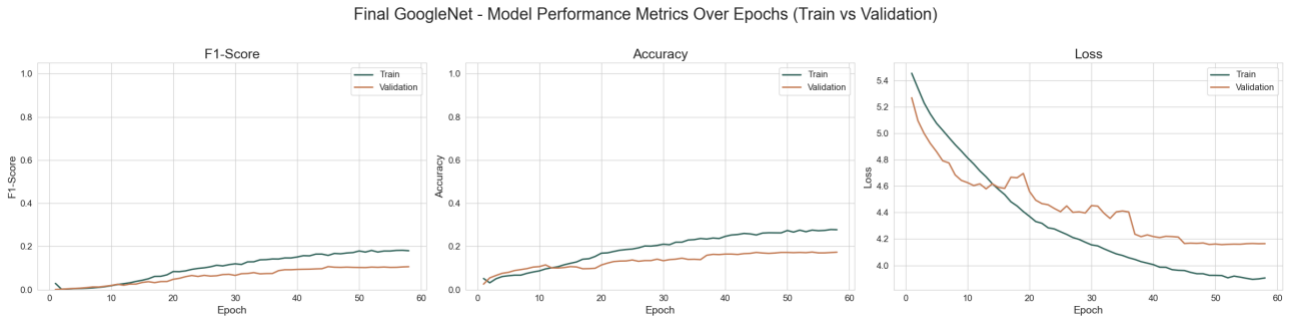


Figure 5 - Training and validation performance of the best-performing GoogLeNet model over 60 epochs

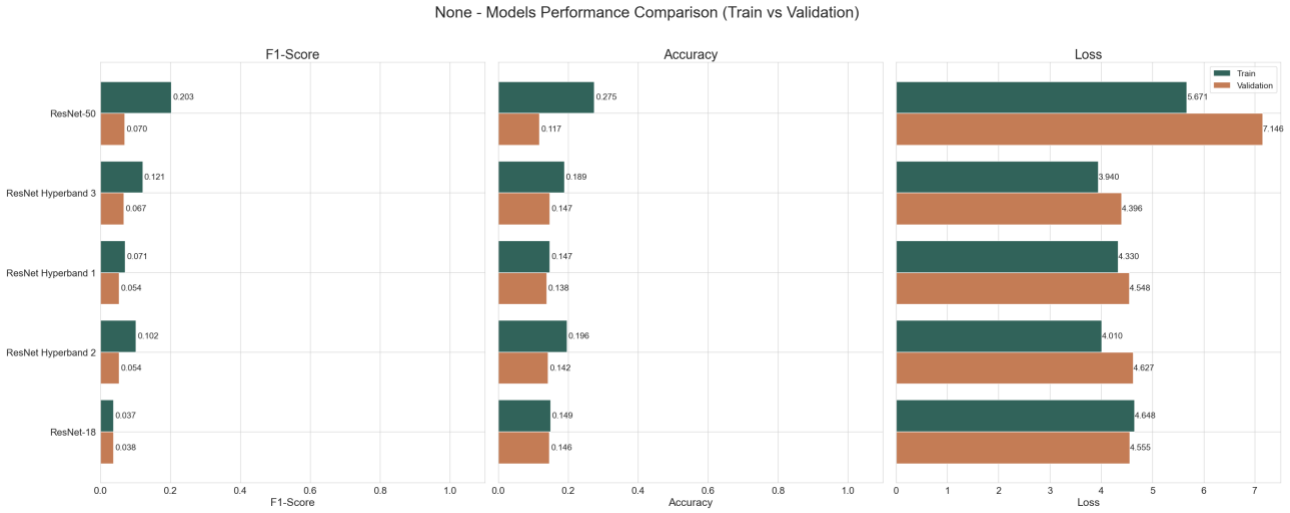


Figure 6 - Visual comparison of the performance metrics across ResNet models

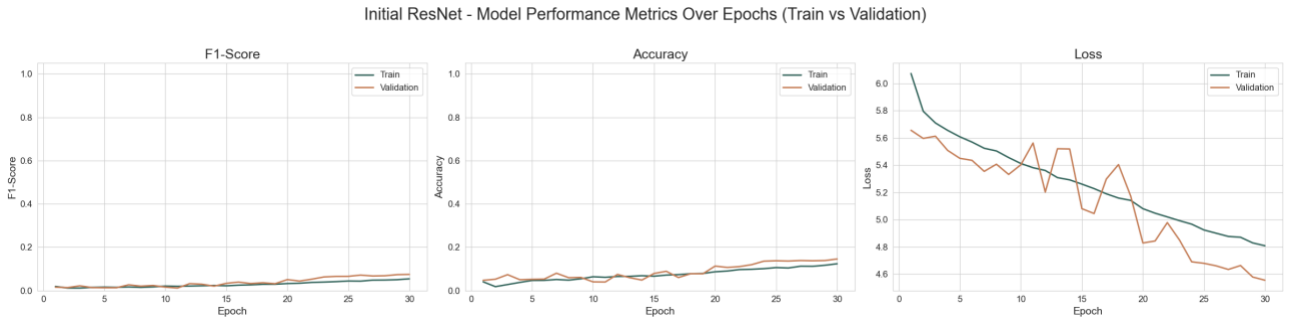


Figure 7 - Training and validation performance of the original ResNet model (also the best-performing) over 30 epochs

Figure 8 - Visual comparison of the performance metrics across DenseNet models

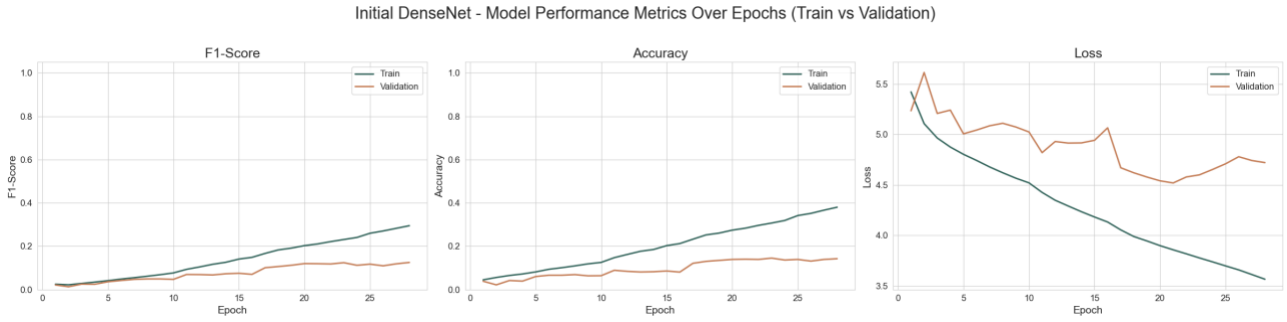


Figure 9 - Training and validation performance of the original DenseNet model over 30 epochs

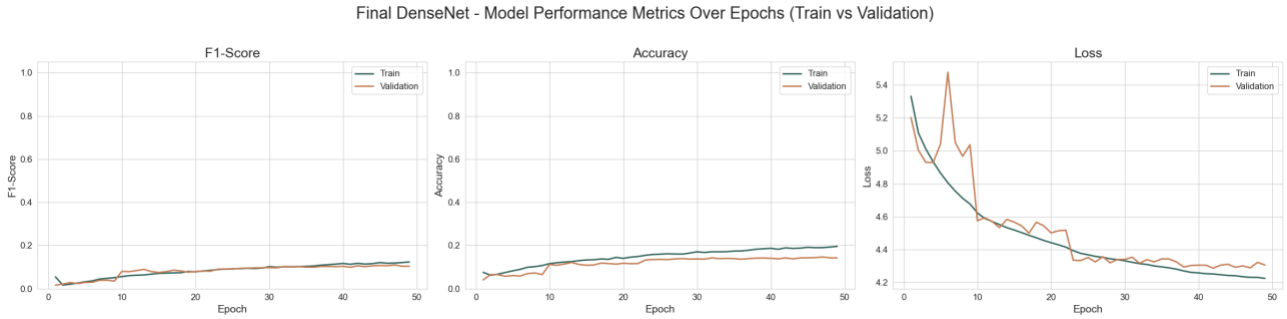


Figure 10 - Training and validation performance of the best-performing DenseNet model over 50 epochs

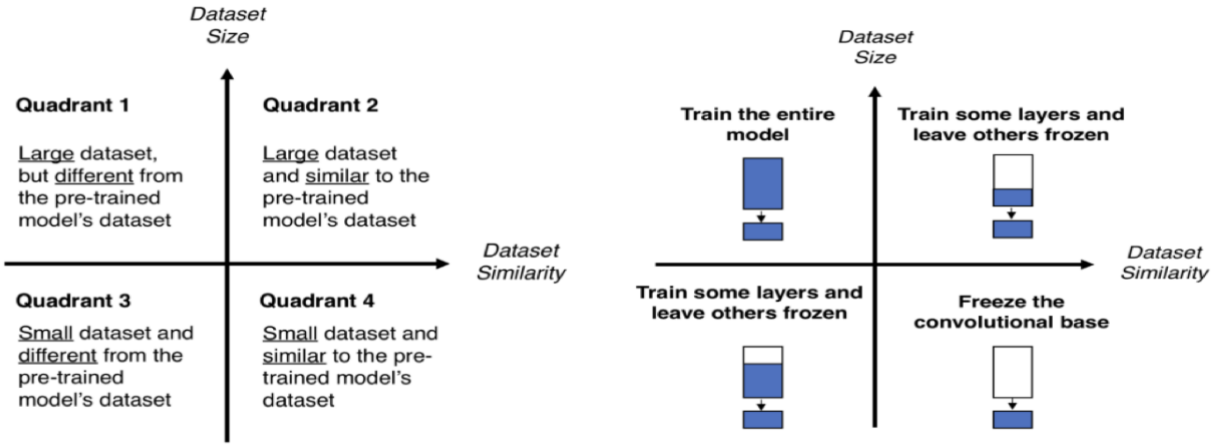


Figure 11 - Size-similarity matrix and decision map for fine tuning pre-trained models

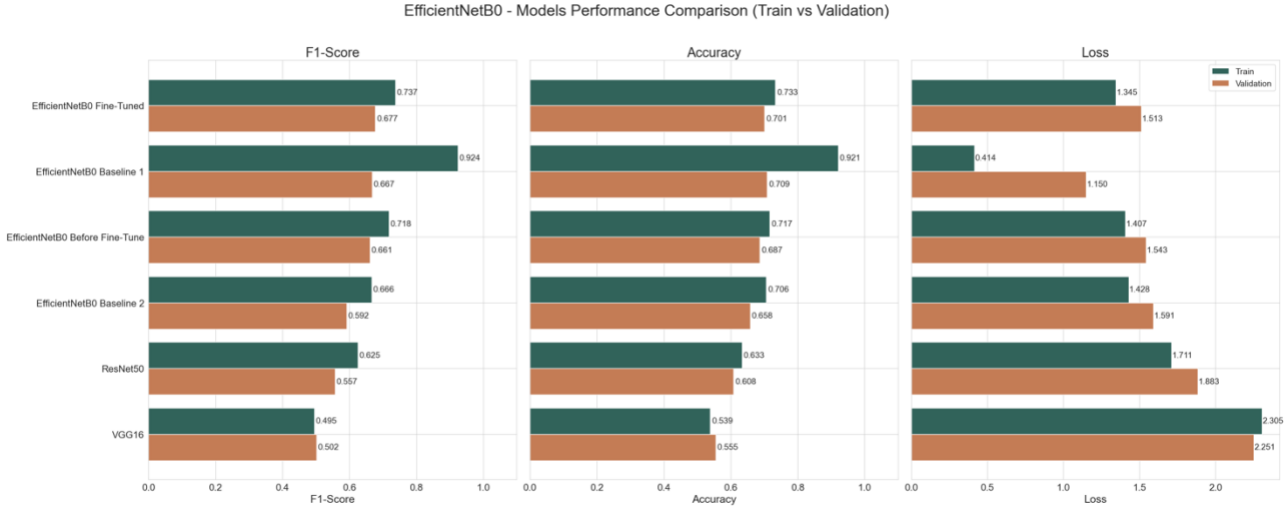


Figure 12 - Visual comparison of the performance metrics across pre-trained models

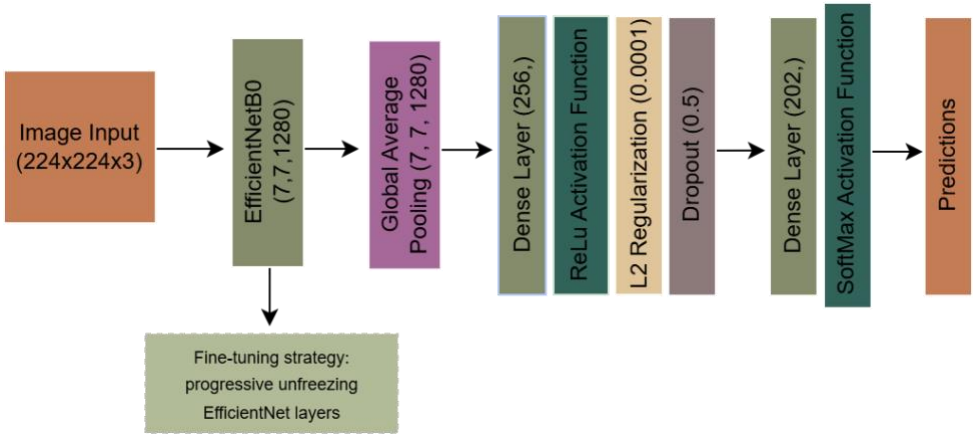


Figure 13 – Diagram for the final model

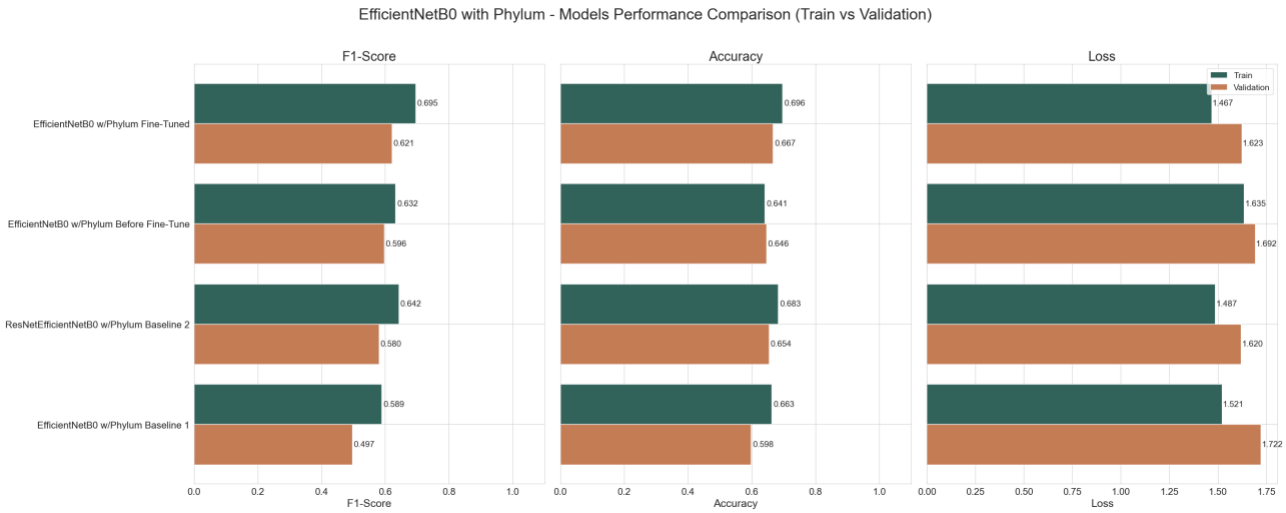


Figure 14 - Visual comparison of the performance metrics across pre-trained EfficientNetB0 with phylum

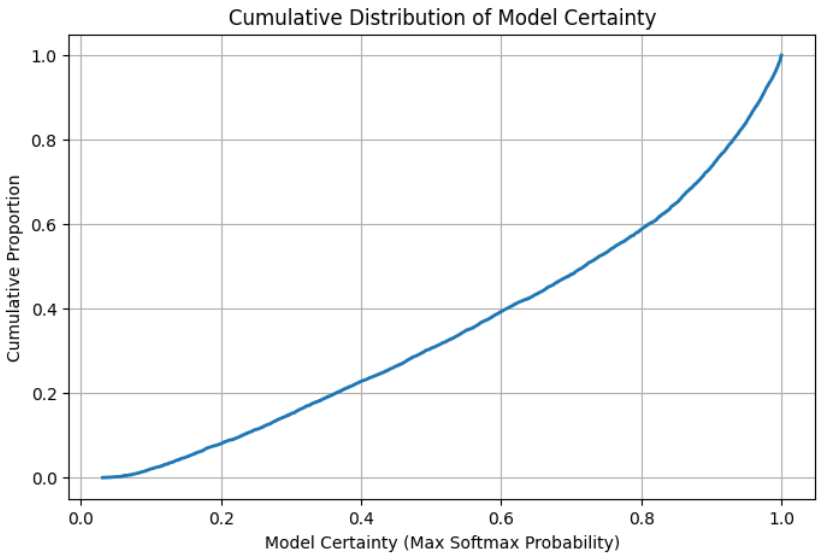


Figure 15 - Cumulative Distribution of Model Certainty of the final model

Figure 16 - Less Confident Misclassifications

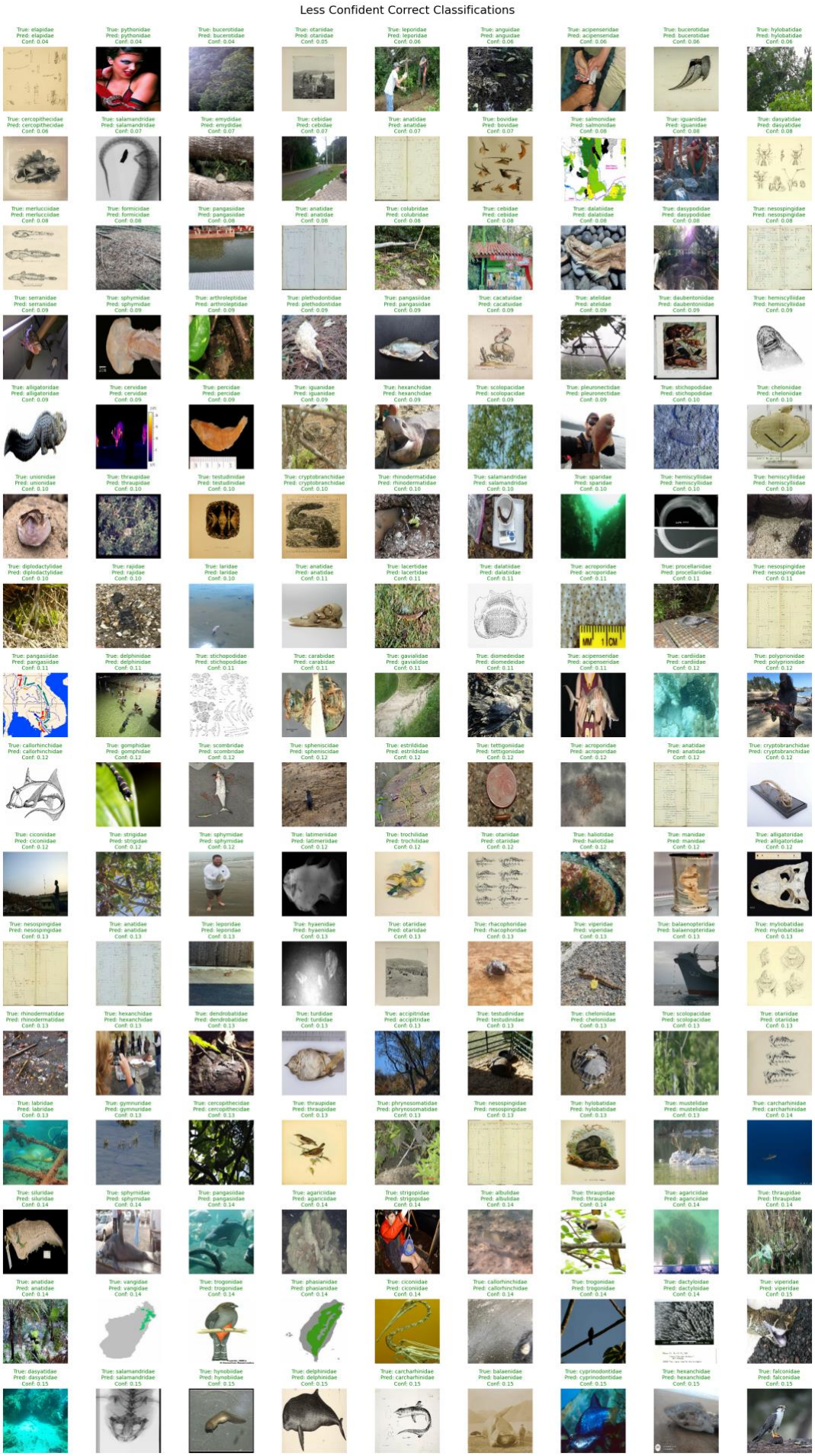


Figure 17 - Less Confident Correct Classifications

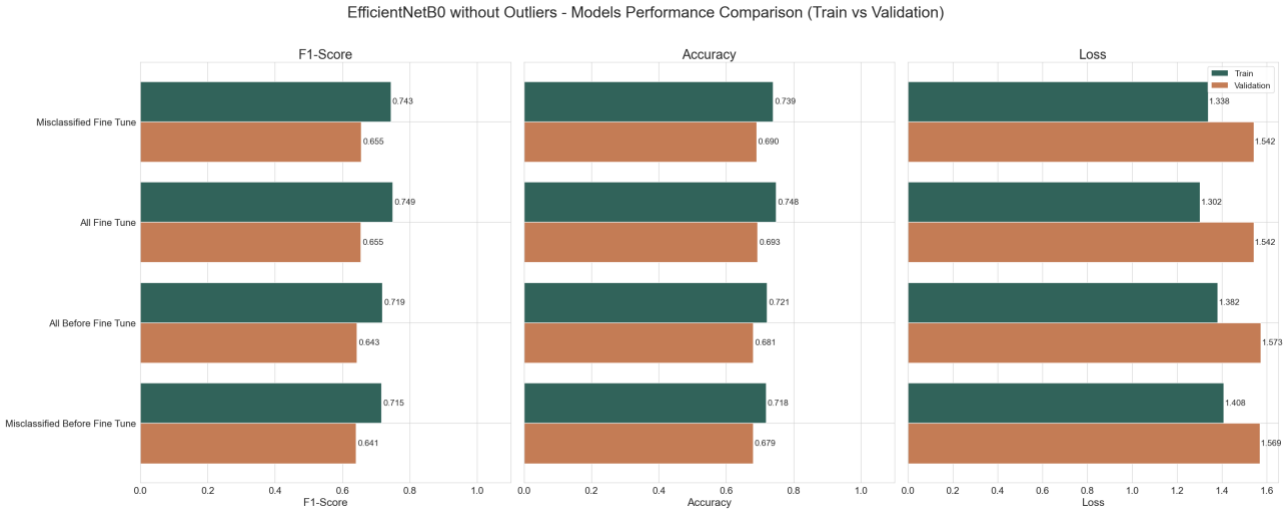


Figure 18 - Visual comparison of the performance metrics across EfficientNetB0 models before and after fine-tuning, following outlier removal ("All" refers to the removal of both correctly and incorrectly classified samples with confidence below 0.1; models without the "All" label exclude only misclassified samples below this threshold)

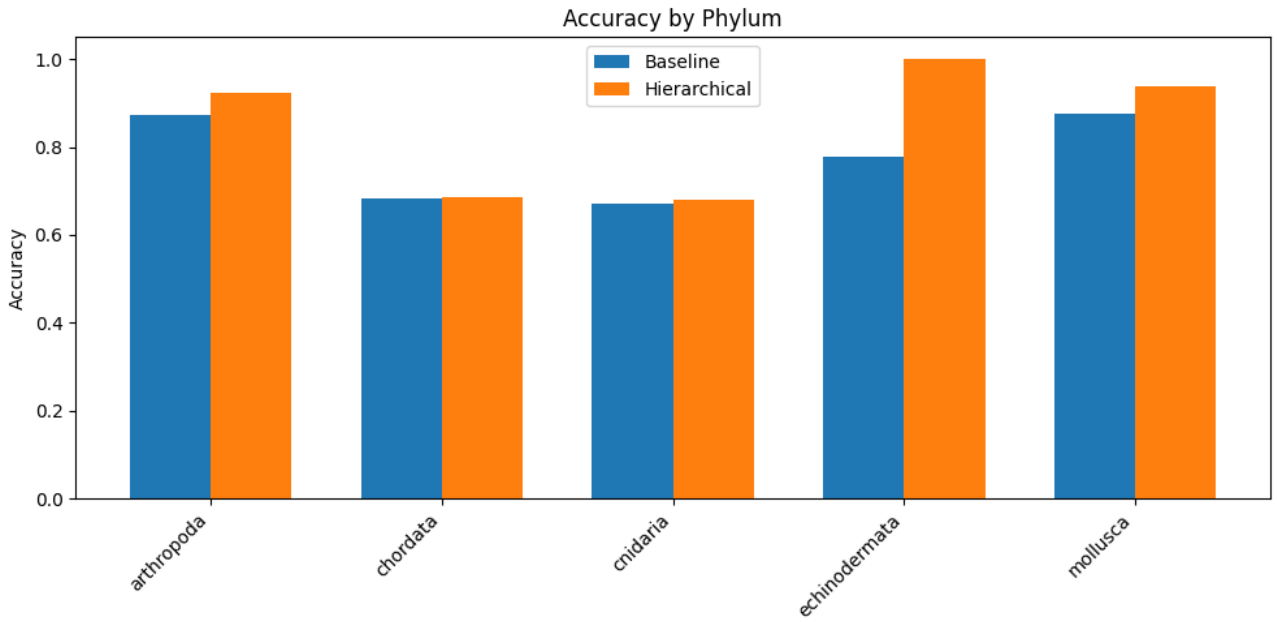


Figure 19 - Accuracy by phylum with and without hierarchical (phylum-based) taxonomy constraints

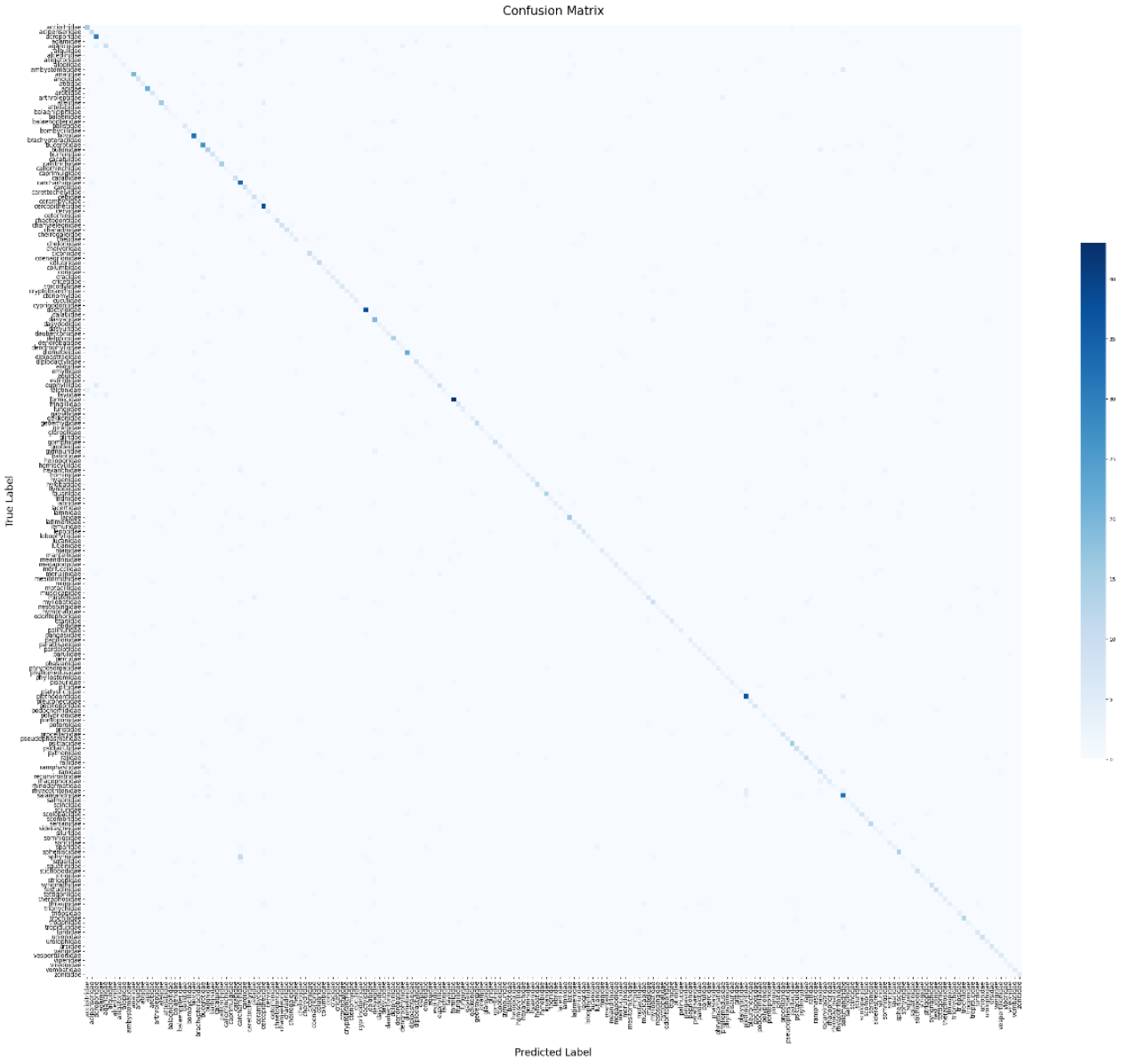


Figure 20 – Confusion Matrix of final model

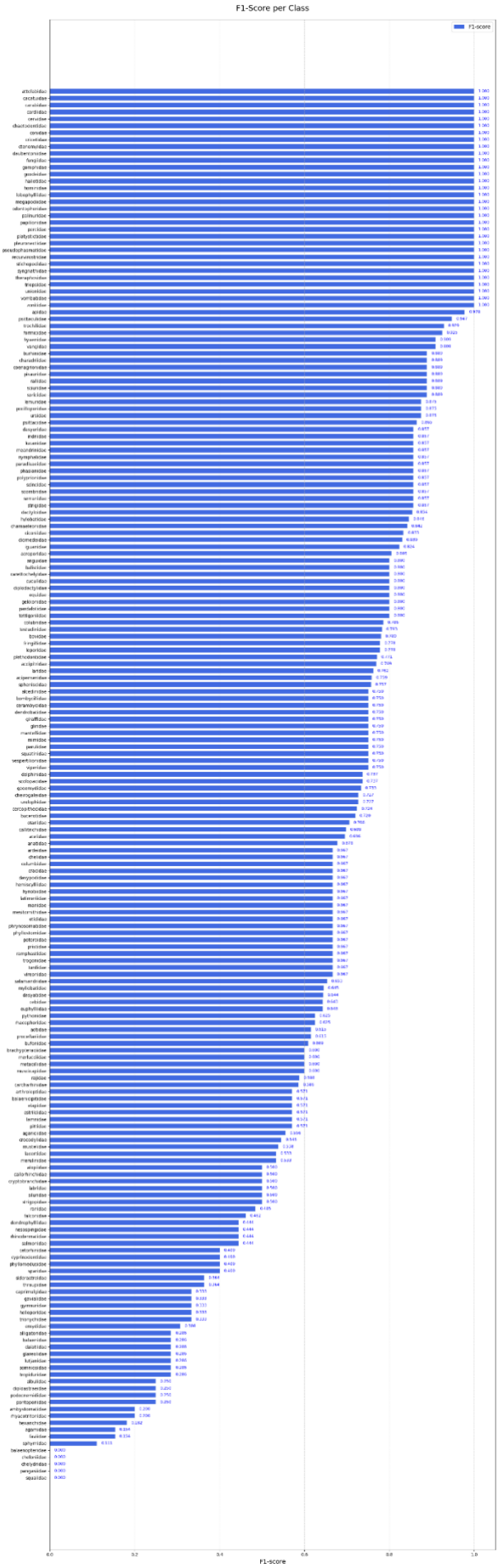


Figure 21 – F1-score per Class of the final model

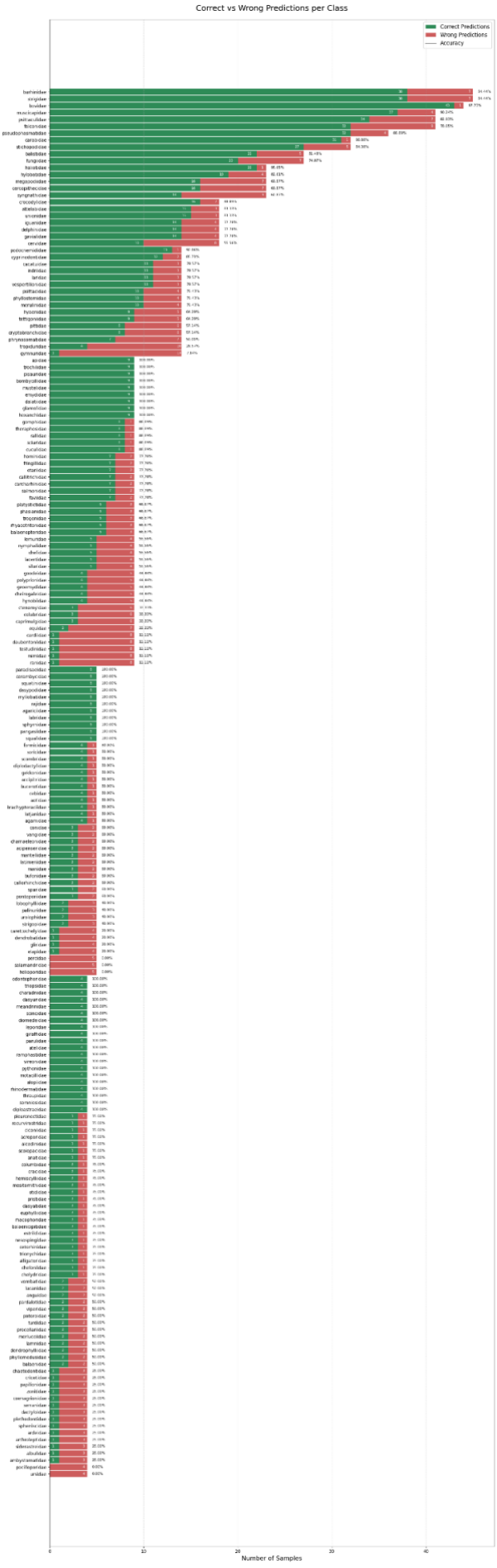


Figure 22 – Accuracy per Class of the final model

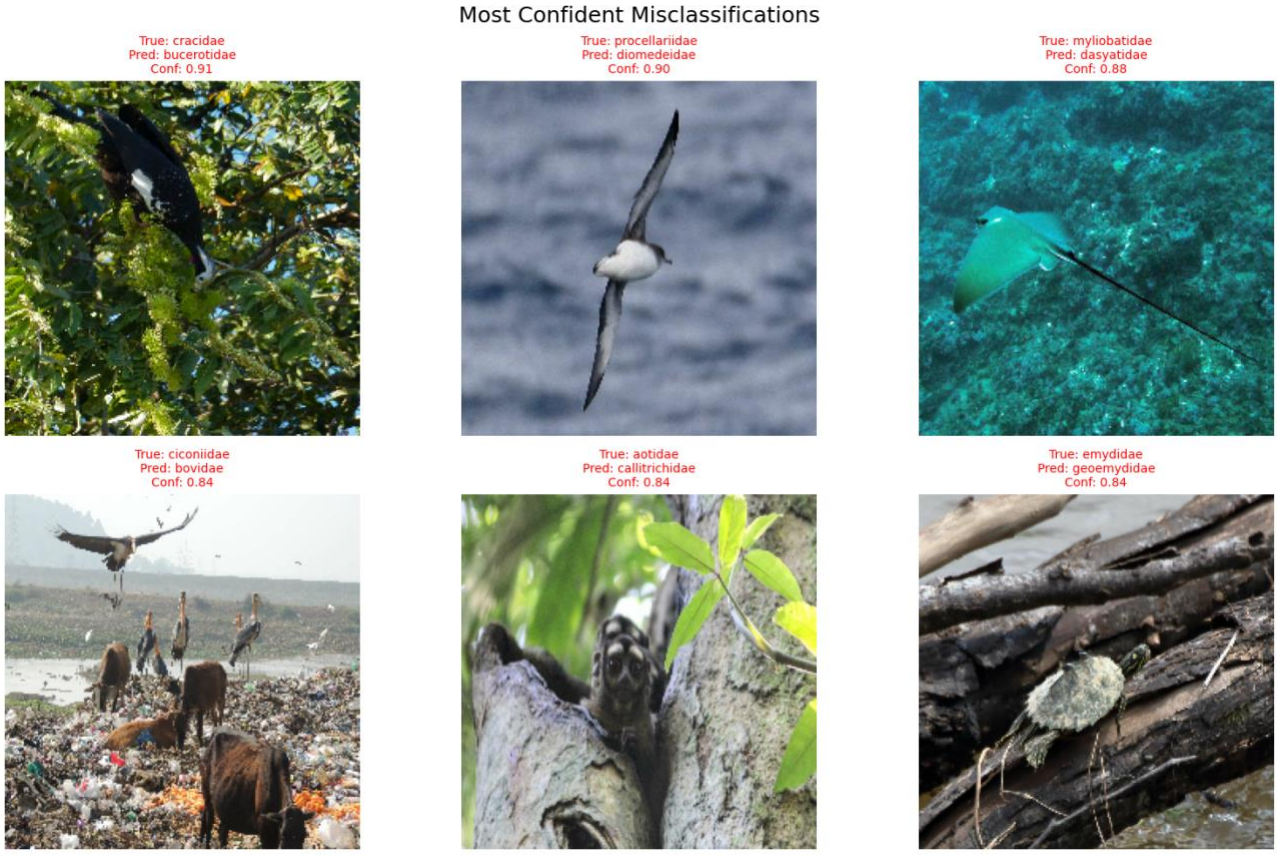


Figure 23 – Most Confident Misclassifications

Figure 24 – Most Confident Correct Classifications

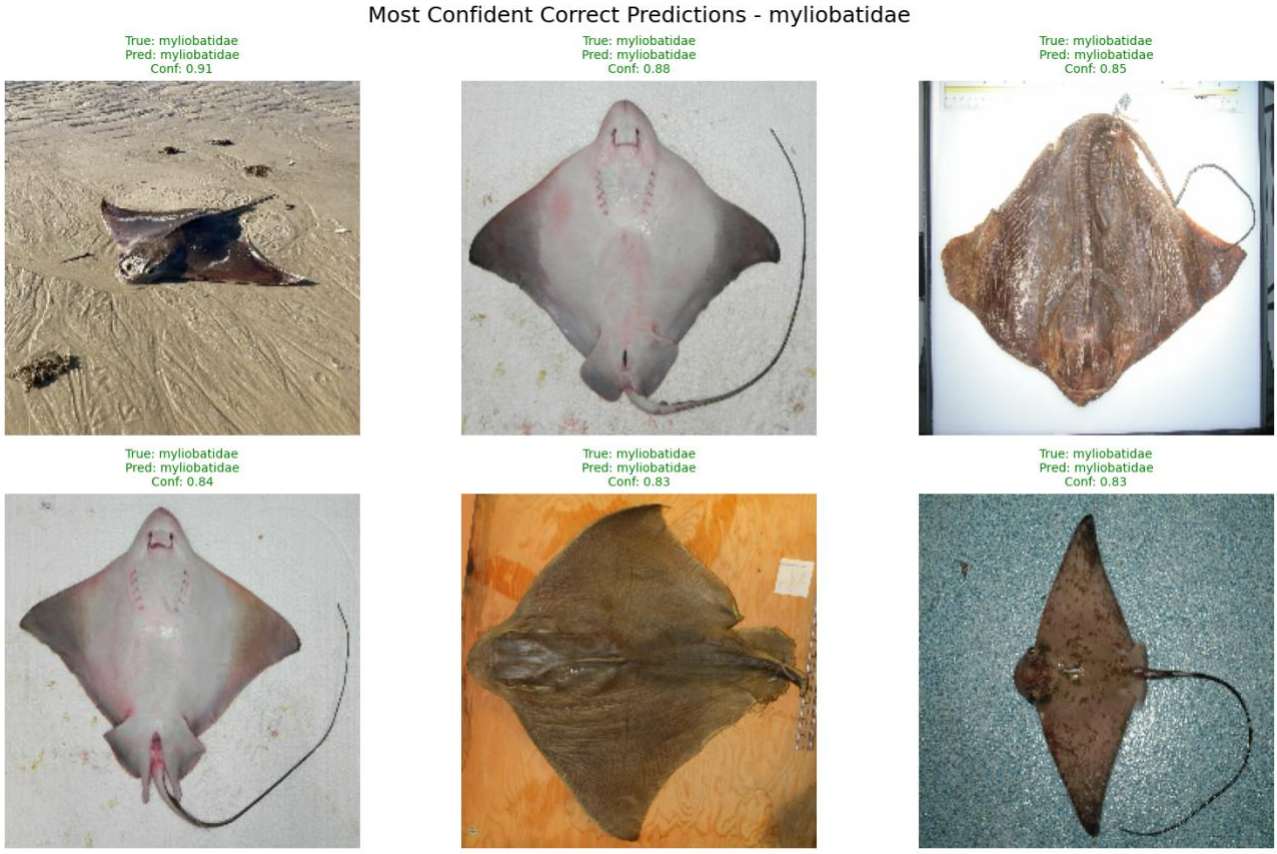


Figure 25 – Most Confident Correct Predictions – myliobatidae

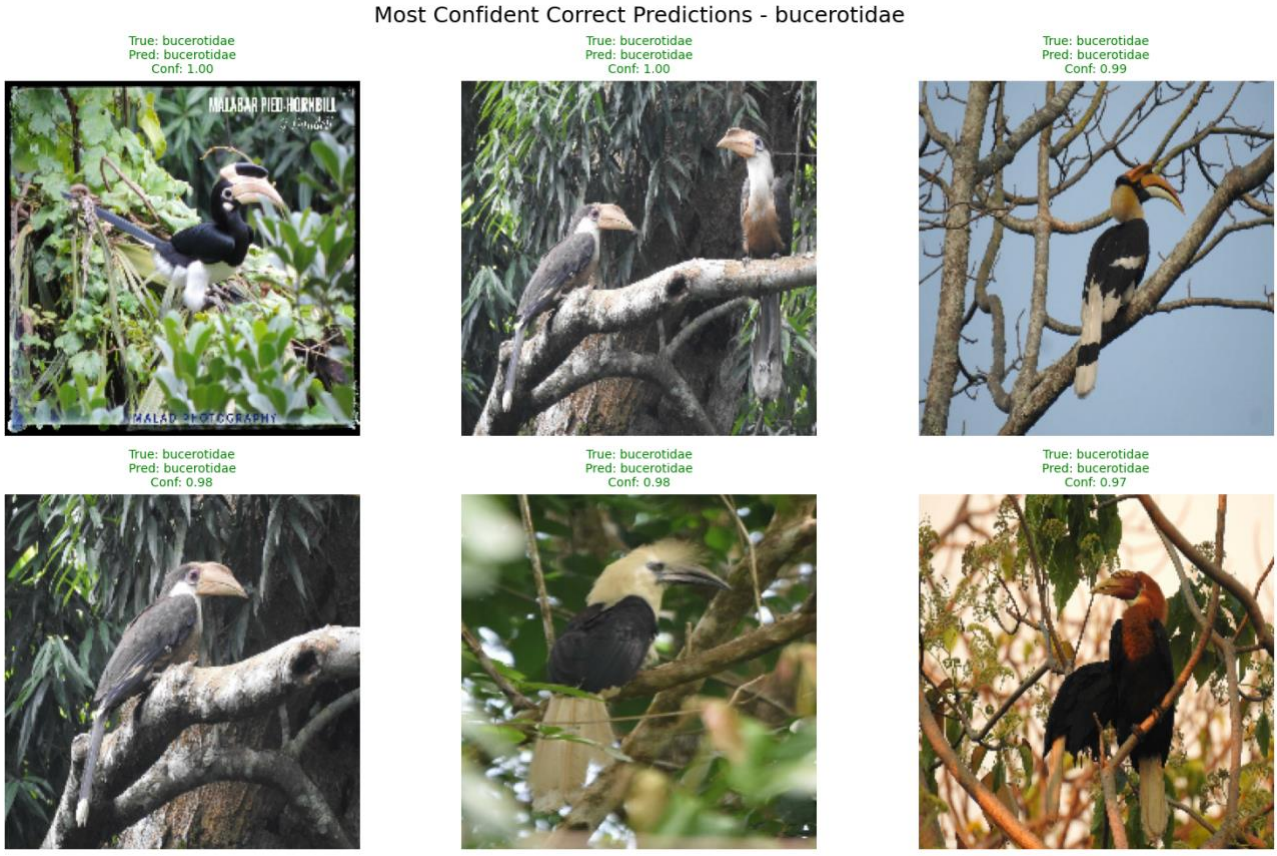


Figure 26 – Most Confident Correct Predictions - bucerotidae

Figure 27 – Most Confident Correct Predictions - apidae

ANNEX B: TABLES

Augmentations	Description
Light	Introduces small perturbations to the image. Includes horizontal flips (50% probability), slight rotations up to $\pm 18°$, zoom within $\pm 5\%$, contrast within $\pm 1\%$, sharpness ($\pm 20\%$), slight changes in brightness, saturation ($\pm 5\%$) and hue ($\pm 1\%$).
Medium	Introduces moderate perturbations to the image Includes horizontal flips (50% probability), rotations up to $\pm 36°$, translations up to $\pm 5\%$, zoom within $\pm 10\%$, contrast within $\pm 15\%$, sharpness ($\pm 30\%$), brightness ($\pm 10\%$), saturation ($\pm 20\%$) and hue ($\pm 2\%$).
Heavy	Uses transformations with stronger effects on image geometry and appearance. Includes horizontal and vertical flips (50% probability), rotations up to $\pm 54°$, translations up to $\pm 10\%$, zoom within $\pm 20\%$, contrast within $\pm 30\%$, sharpness ($\pm 40\%$), brightness ($\pm 20\%$), saturation ($\pm 30\%$) and hue ($\pm 5\%$).
Grayscale	Converts RGB images to grayscale and back to RGB, followed by random contrast adjustment (up to 40%). Designed to reduce model reliance on color information.
Grayscale Plus	Builds on the previous augmentation by adding random flips, zoom, rotation, and sharpness. Encourages the model to focus on shape and texture by reducing reliance on color information.
Randaugment	Randomly selects and applies a fixed number of transformations per image from a predefined pool, controlled by magnitude.
MixUp	Creates new training samples by mixing two images and their labels using linear interpolation drawn from a Beta distribution ($\alpha = 0.2$). For more details, consider Annex C: Definitions.
Geometric Transformations	Focuses exclusively on geometric transformations: horizontal and vertical flips, moderate rotations, zoom, and translation ($\pm 10\%$).
Color Lightening	Focuses exclusively on color transformations, such as brightness ($\pm 10\%$), contrast ($\pm 15\%$), saturation ($\pm 20\%$), and hue ($\pm 2\%$).

Table 1 - Description of Augmentation Strategies Used in Training

architecture	train_loss	val_loss	train_accuracy	val_accuracy	train_f1_macro	val_f1_macro
baseline	0.0289	8.4305	0.9921	0.1697	0.9974	0.1326
vgg_model	1.0658	5.0694	0.8401	0.1575	0.8509	0.0978
dense_net	4.3609	4.7914	0.1454	0.1185	0.0891	0.0573
googlenet	4.0019	4.3110	0.1400	0.1297	0.0657	0.0492
resnet	4.5899	4.7907	0.1333	0.1085	0.0567	0.0362

Table 2 – Performance metrics for the five architectures trained with oversampling and no data augmentation, evaluated after 20 epochs

augmentation	train_loss	val_loss	train_accuracy	val_accuracy	val_f1_macro
none	1.3911	4.5692	0.9362	0.1920	0.1255
mixup	3.7449	4.7442	0.4610	0.1642	0.0930
randaugment	3.4322	4.5227	0.3413	0.1697	0.0846
color_lightening	3.3282	4.6069	0.3733	0.1720	0.0789
geometric_transformations	4.0121	4.5991	0.2145	0.1586	0.0755
heavy	3.6324	4.5743	0.2922	0.1469	0.0638
grayscale_plus	2.9042	4.9170	0.5308	0.1230	0.0487
medium	4.5955	4.7878	0.1363	0.1319	0.0420
light	5.7216	5.6504	0.0232	0.0323	0.0014

Table 3 – Evaluation of augmentation strategies on ResNet with oversampling, after 30 epochs

Model	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score
GoogLeNet Original	3.2043	3.9341	0.4379	0.2092	0.3435	0.1359
GoogLeNet Modified	3.6404	3.9415	0.3179	0.1892	0.2212	0.1095

Table 4 – Performance comparison between the original and modified GoogLeNet architectures, trained with EarlyStopping and ReduceLROnPlateau for 100 epochs

architecture	train_loss	val_loss	train_accuracy	val_accuracy	train_f1_macro	val_f1_macro
resnet50	5.6706	7.1465	0.2746	0.1174	0.2033	0.0697
resnet18	4.6479	4.5547	0.1492	0.1458	0.0372	0.0379

Table 5 – Performance comparison between the original and modified ResNet architectures, trained with EarlyStopping and ReduceLROnPlateau for 30 epochs

Model	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score
DenseNet Original	4.9534	4.9232	0.0947	0.0460	0.0952	0.0330
DenseNet Simplified	4.2634	4.5188	0.2091	0.1665	0.1402	0.0825

Table 6 – Performance comparison between the original and modified DenseNet architectures, trained with EarlyStopping and ReduceLROnPlateau, for 30 epochs

Parameter	*Search Space*
dropout_rate	0.2 to 0.5 (step = 0.1)
Filters_init	{32, 64, 128}
neurons_dense	{128, 256, 512, 1024}
kernel_size	{5, 7, 9}
label_smoothing	0.0 to 0.1 (step = 0.01)
learning_rate	{1e-4, 5e-4}

Table 7 – Hyperparameter search space used in Hyperband tuning for GoogLeNet

Parameter	*Search Space*
l2_weight	{1e-5, 1e-4, 5e-4}
head_units	{128, 256, 512}
head_activation	{"relu", "leaky_relu"}
dropout_rate	{0.4, 0.5}
blocks_stage1	{2, 3}
blocks_stage2	{2, 3, 4}
blocks_stage3	{2, 4, 6}
blocks_stage4	{2, 3}

Table 8 – Hyperparameter search space used in Hyperband tuning for ResNet

Parameter	*Search Space*
growth_rate	{8, 12, 16}
compression	{0.4, 0.5, 0.6}
dropout_rate	0.2 to 0.6 (step = 0.1)
learning_rate	{1e-4, 5e-4}
label_smoothing	{0.0, 0.05, 0.1}

Table 9 – Hyperparameter search space used in Hyperband tuning for DenseNet

Model	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score
GoogLeNet Hyperband 1	3.8582	4.1567	0.2888	0.1714	0.1971	0.1023
GoogLeNet Hyperband 2	3.6369	4.0851	0.3784	0.1976	0.2797	0.1236
GoogLeNet Hyperband 3	4.6765	4.5372	0.1260	0.1202	0.0403	0.0350

Table 10 – Performance of three GoogLeNet variants selected through Hyperband tuning, trained with EarlyStopping and ReduceLROnPlateau for 100 epochs

Model	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score
ResNet Hyperband 1	4.3298	4.5480	0.1470	0.1380	0.0707	0.0540
ResNet Hyperband 2	4.0103	4.6275	0.1960	0.1425	0.1020	0.0537
ResNet Hyperband 3	3.9396	4.3959	0.1887	0.1469	0.1209	0.0671

Table 11 – Performance of three ResNet variants selected through Hyperband tuning, trained with EarlyStopping and ReduceLROnPlateau for 100 epochs

Model	Train Loss	Validation Loss	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score
DenseNet Hyperband 1	3.9695	4.3102	0.2500	0.1491	0.1611	0.0792
DenseNet Hyperband 2	4.2234	4.3047	0.1962	0.1425	0.1003	0.0674
DenseNet Hyperband 3	4.7513	4.8496	0.1185	0.1085	0.0434	0.0350

Table 12 – Performance of three DenseNet variants selected through Hyperband tuning, trained with EarlyStopping and ReduceLROnPlateau for 100 epochs

Model	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score	Train Loss	Validation Loss
EfficientNetB0 Fine-Tuned	0.733482	0.700612	0.736858	0.677013	1.344867	1.513090
EfficientNetB0 Baseline 1	0.920720	0.708959	0.923846	0.667322	0.414008	1.149756
EfficientNetB0 Before Fine-Tune	0.716607	0.686700	0.717598	0.660923	1.406863	1.542795
EfficientNetB0 Baseline 2	0.706366	0.657763	0.666257	0.591676	1.428393	1.591309
ResNet50	0.633125	0.608236	0.625082	0.556639	1.710753	1.883143
VGG16	0.538839	0.554814	0.494909	0.501549	2.305127	2.251417

Table 13 - Training and validation performance metrics of pre-trained models

Model	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score	Train Loss	Validation Loss
EfficientNetB0 w/Phylum Fine-Tuned	0.6964	0.6667	0.6950	0.6209	1.4667	1.6233
EfficientNetB0 w/Phylum Before Fine-Tune	0.6413	0.6461	0.6324	0.5961	1.6352	1.6921
ResNetEfficientNetB0 w/Phylum Baseline 2	0.6829	0.6539	0.6421	0.5800	1.4866	1.6197
EfficientNetB0 w/Phylum Baseline 1	0.6631	0.5977	0.5887	0.4966	1.5212	1.7217

Table 14 - Training and validation performance metrics of EfficientNetB0 using phylum

Model	Train Accuracy	Validation Accuracy	Train F1-Score	Validation F1-Score	Train Loss	Validation Loss
Misclassified Fine Tune	0.739399	0.690039	0.743414	0.655489	1.338020	1.541555
All Fine Tune	0.747748	0.692821	0.748861	0.654693	1.302281	1.541790
All Before Fine Tune	0.721378	0.680579	0.718699	0.643141	1.381631	1.573125
Misclassified Before Fine Tune	0.717920	0.679466	0.715404	0.640639	1.408023	1.568748

Table 15 - Training and validation performance metrics of EfficientNetB0 before and after fine-tuning, following outlier removal ("All" refers to the removal of both correctly and incorrectly classified samples with confidence below 0.1; models without the "All" label exclude only misclassified samples below this threshold)

ANNEX C: DEFINITIONS

1. MixUp

MixUp, introduced by Zhang et al. (2018) [Bib. 13], is a data augmentation technique based on the principle of generating synthetic training examples through linear interpolations of pairs of inputs and their corresponding labels. Formally, given two examples (x_i, y_i) and (x_j, y_j) drawn at random from the training set, MixUp constructs a new sample $(\tilde{x}, \tilde{y})$ as follows:

$$\tilde{\boldsymbol{x}} = \boldsymbol{\lambda x_i} + (\mathbf{1} - \boldsymbol{\lambda})\boldsymbol{x_j}$$

$$\tilde{\boldsymbol{y}} = \boldsymbol{\lambda y_i} + (\mathbf{1} - \boldsymbol{\lambda})\boldsymbol{y_j}$$

Here, $\lambda \in [0,1]$ is a mixing coefficient sampled from a Beta distribution:

$$\lambda \sim Beta(\alpha, \alpha)$$

Where $\alpha > 0$ is a hyperparameter that controls the strength of interpolation.

The choice of the Beta distribution is motivated by its flexibility in generating values within the $[0, 1]$ interval and its ability to model symmetric or asymmetric interpolations depending on the parameter α.

When $\alpha = 1$, the distribution is uniform, for $\alpha < 1$, it favors samples close to 0 or 1 (stronger bias toward one sample); and for $\alpha > 1$, it yields more balanced mixtures. This property allows precise control over how much one sample dominates the interpolation, thus enabling the regularization of decision boundaries during training.

MixUp has been shown to improve generalization and calibration of deep neural networks across various classification tasks, particularly in scenarios with label noise or class imbalance. The original study demonstrated empirical benefits on several benchmark datasets. The technique introduces convex combinations in both the input and label spaces, thereby enforcing linear behavior between samples and reducing overfitting.

2. VGGNet

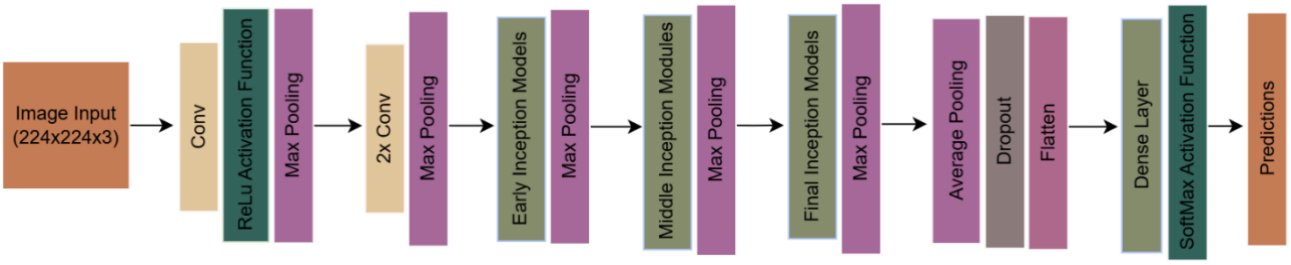
VGGNet is an architecture introduced by the Visual Geometry Group at Oxford in 2014. This architecture only uses 3x3 convolutional layers and 2x2 max pooling and ends with 2 or 3 fully connected layers before softmax.

3. GoogleNet

GoogleNet architecture was introduced by Google in 2014, and it is based in inception modules, that process input with multiple filter sizes in parallel and concatenate their outputs. Each inception model applies the following operations to the inputs:

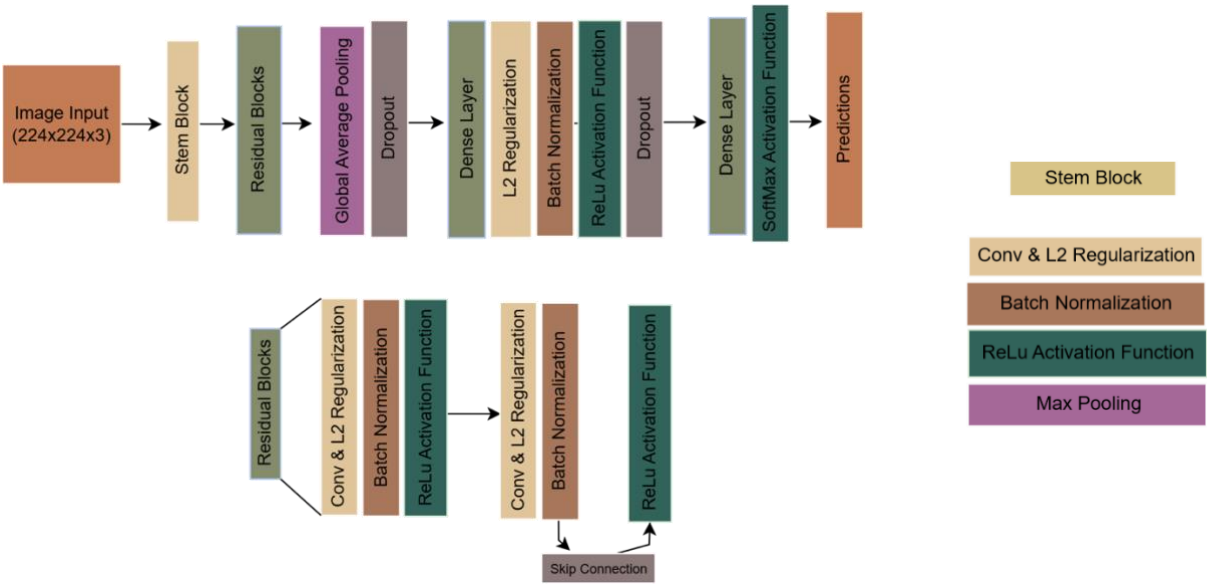
- 1×1 convolutions for dimensionality reduction
- 3×3 and 5×5 convolutions for spatial feature extraction
- 3x3 max pooling followed by 1x1 convolutions in the pooling path
- All outputs are after concatenated to form the module's output.

This model is more computationally efficient than models like VGGNet since it uses a significant reduced number of parameters.



4. ResNet

Residual Networks (ResNet) was introduced originally by He et al. (2016). This architecture is based on residual blocks and has a key innovation – skip connections - that address the vanishing gradient problem in deep neural networks. Each residual block has multiple convolutional layers followed by batch normalization and ReLu activation. The skip connections allow the network to bypass certain layers, which allows gradients to flow directly through the network.



5. DenseNet

In DenseNet architecture proposed by **Huang et al. (2017), e**ach layer within the same dense block receives as input feature maps from all preceding layers. This encourages gradient flow, reduces the number of parameters and allows each layer to access a more substantial set of features.

Key parameters in DenseNet are the following:

- Layers per block: Number of layers in each dense block
- Growth-Rate: Number of new feature maps each layer contributes.
- Compression: Applied in transition layers to reduce the number of feature maps. This parameter can help reduce model size and overfitting.

The diagram below further clarifies the architecture of DenseNet and other relevant details, such as activation functions and pooling techniques.

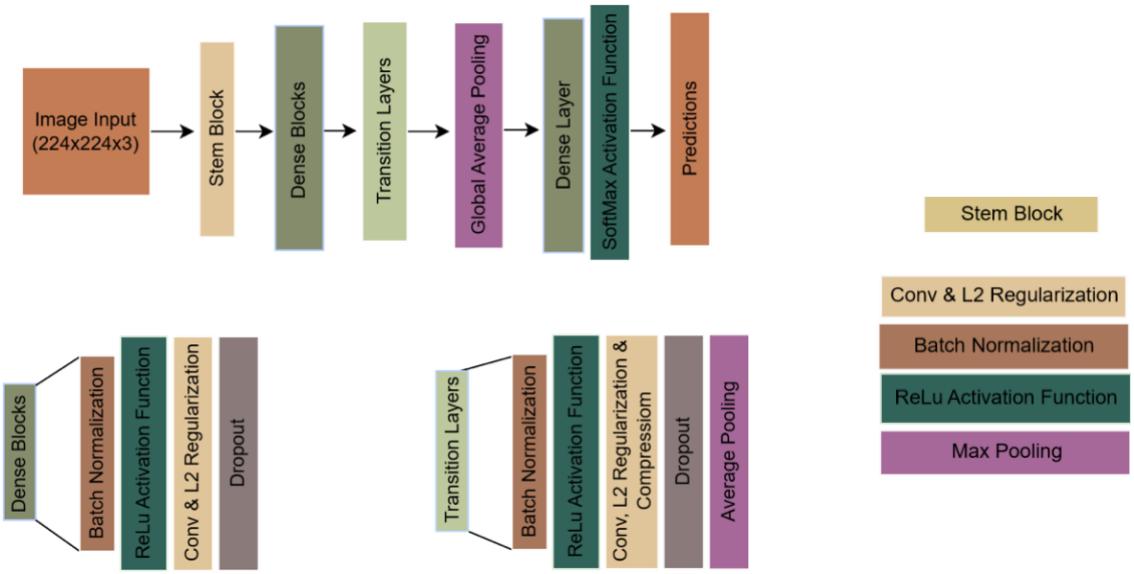


Image Input
(224x224x3)
Stem Block
Dense Blocks
Transition Layers
Global Average Pooling
Dense Layer
SoftMax Activation Function
Predictions
Dense Blocks
Batch Normalization
ReLu Activation Function
Conv & L2 Regularization
Dropout
Transition Layers
Batch Normalization
ReLu Activation Function
Conv, L2 Regularization & Compression
Dropout
Average Pooling
Stem Block
Conv & L2 Regularization
Batch Normalization
ReLu Activation Function
Max Pooling