Deep Learning Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Deep Learning for Classification of Clinical Skin Disease Images

Group 24

André Lopes, 20230570

Catarina Silva, 20230368

Eugénia Rosário, 20220598

Luís Queiroz, 20230584

Pedro Cerejeira, 20230442

April 2024

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

INDEX

1. Introduction

Dermatology presents a significant challenge in medical diagnostics due to the visual complexity and variability of skin conditions. This report aims to outline the development of a deep-learning model tailored for a dermatology classification task. Our methodology focuses on creating a deep-learning model capable of generalizing well to unseen data, thereby facilitating robust dermatological classification.

The project centers on a dataset comprising various images, each annotated with disease details such as malignancy, benignity, angle scores, and URLs linking to corresponding skin lesion images. The main objective for classification revolves around the "label" column, found within the provided CSV file.

This dataset is sourced from the “FITZPATRICK17” repository, which incorporates data from the "DermaAmin" and "ATLAS Dermatologico" collections. With a total of 114 distinct skin conditions represented, our dataset comprises a diverse array of cases, with each condition having between 53 to 653 images.

2. Exploratory Analysis

The dataset under consideration encompasses 16,577 rows with 114 unique labels. Each row is uniquely identified by an MD5 hash, ensuring data integrity. The column "fitzpatrick_scale" delineates skin color on a scale from 1 to 6, sourced from "Scale AI," with some entries marked as -1, presumably indicating unclassified instances. Similarly, the "fitzpatrick_centaur" column, obtained from "Centaur Lab," reflects skin color on the same scale, with -1 values representing unclassified instances.

Additionally, the dataset includes columns categorizing skin diseases into nine and three partitions, as illustrated in the provided images. For instance, 'nine_partition_label' delineates diseases into nine distinct categories such as inflammatory, malignant epidermal, and benign dermal, whereas 'three_partition_label' offers a broader classification into non-neoplastic, malignant, and benign categories.

Concerning data quality, the “qc” column provides insights into observations regarding label validity, with varying degrees of diagnostic certainty ranging from Diagnostic to Potentially, as well as instances of wrongly labeled or characteristic observations.

Of note, the “md5hash” column, while essential for integrity checking, holds no predictive modeling or statistical significance; therefore, we've opted to exclude it from further consideration. Additionally, concerning missing values, the “qc” column shows 16,073 missing entries, accounting for 96.96% of its total. Similarly, the “url” column has 41 missing values, representing merely 0.25% of its entries. Accordingly, these discrepancies required additional attention and were subsequently addressed.

3. Data Split

Regarding data splitting, we initially designated the target variable as the ‘label’ column, representing the various skin conditions, and excluded it from the data set. Subsequently, we divided the dataset into training, validation, and test sets, allocating 70% for training, 15% for validation, and 15% for testing. Stratification based on the target variable was utilized to mitigate biases stemming from class imbalances, thus ensuring uniform label distributions across subsets.

4. Data Loading and Preprocessing

4.1. Missing Values

In our dataset, the 'qc' column displayed over 96% NaN values. We determined that these were not true missing values; instead, their absence signified that there were no errors in the labeling of these records. Consequently, we replaced the NaN values with 'No Issue' to accurately reflect this finding. For the 'url' column, which also contained missing values, we found no feasible method to impute or treat these absences.

Therefore, we decided the most appropriate action was to remove these records. Given that these rows constitute only 0.25% of the dataset, their removal is expected to have a minimal impact on our analysis.

4.2. Image Loading

Efficient handling of large image datasets is essential for the success of deep learning models, particularly due to the computational and memory constraints they encounter. To address this, we built a Data Generator approach for dynamic image loading and processing. We have also developed an automated script to facilitate the download and organization of images into appropriate directories for training, validation, and testing.

4.2.1. Image Downloading Strategy

In this phase, we set up a structured directory system where images are organized by dataset category - train, validation, and test - and further sorted by class labels. This organization supports seamless integration with Keras' ImageDataGenerator, enabling automated labeling during model training.

4.2.2. Implementation Details

The image downloading process involves iterating over a DataFrame that contains URLs and class labels of images. Each image is systematically saved in a folder corresponding to its class label, ensuring each image is easily accessible and correctly categorized for training purposes.

4.3. Image Normalization and Resizing

As part of our preprocessing pipeline, we used some strategies for resizing and normalizing the images and therefore ensuring that all are suitable for processing our models.

4.3.1. Image Resolution Analysis

To identify the most appropriate resizing strategy, we started by determining the common default resolutions of images in our dataset. This involved sampling a small number of URLs from each dataset split (training, validation, test) and fetching the images to record their width and height. This method enabled us to observe a wide range of image dimensions, and based on that observation we made the decision to set a standard resolution for all images.

The analysis indicated that while average widths and heights varied, the most common resolutions were significantly lower than the maximum dimensions found. Considering computational efficiency and the need to capture sufficient detail for accurate model training, we settled on a target size of 256x256 pixels. This size is widely recognized as optimal for many CNN architectures, providing a good balance between input detail and computational demand.

4.3.2. Resizing Strategies

After analyzing the most common image resolutions, we normalized the image pixels from the standard range of 0-255 to 0-1 by dividing them by 255. This adjustment was crucial for our CNN models, as it ensures that the model inputs have consistent and small numerical values, which aids in accelerating the convergence during training.

5. Data Augmentation

To enhance the robustness and the capacity for generalization of our models, we applied image augmentation during training. This strategy serves two primary purposes: increasing the training dataset size by generating variations of the original images, which helps in preventing overfitting by providing a broader range of training examples; enhancing model reliability by training the model on variations of each image,

including changes in angles, lighting, and scale, we ensure that it does not merely memorize the training data, but instead learn to recognize the underlying patterns.

For data augmentation, we employed different strategies to ensure that the model is exposed to diverse aspects of the data. One such strategy involves the use of TensorFlow's ImageDataGenerator class to perform augmentation during training. Parameters such as rotation range, width, and height shift range, among others, were configured to generate images with random transformations.

Central to our augmentation approach is the center_crop() function, which crops the center of each image to a specified size, maintaining essential features while augmenting the data. By implementing augmentation techniques, we aim to ensure that the model never sees the exact same picture twice during training, thereby promoting better generalization. Additionally, separate data generators were created for training, validation, and testing datasets, with augmentation applied only to the training data. Through these augmentation strategies, our deep learning model gains exposure to a diverse range of image variations, ultimately leading to improved performance in dermatology classification tasks.

6. Model Testing

Before initiating model testing and development, we established two functions to handle both saving and loading the trained model along with its training history. This allows us to bypass the need for model training every time we conduct our analysis, thereby saving valuable time and ensuring consistent results for each run. Following this preparation, we then proceeded to create and assess the models.

During model testing, we explored a variety of model architectures to assess their effectiveness for this classification task. Our approach, explained with a schema in Figure 5, encompassed the implementation of two distinct types of models: entirely custom-designed models, and pre-trained models with customized layers we added.

Firstly, we developed a custom model architecture comprising convolutional layers for feature extraction and dense layers for classification. This model was labeled as "model_custom". Subsequently, we delved into transfer learning techniques, initially implementing a model based on the VGG-16 architecture pre-trained on the ImageNet dataset. This "model_transfer" approach facilitated quicker training and improved performance by leveraging pre-trained weights. We replaced the top tree fully connected layers with a sequence of layers tailored to our classification task, enhancing the model's adaptability to our specific problem.

Expanding on transfer learning, we introduced "model_transfer_2layers", which retained two of the last three fully connected layers of the VGG-16 architecture. This modification aimed to strike a balance between leveraging pre-trained knowledge and adapting the model to our specific task, thereby potentially enhancing performance. This hypothesis, however, would soon come to turn out false, in our results.

The initial performance of these three models was poor, exhibiting very low weighted F1 scores and showing minimal improvement after just a few epochs (signs of underfitting). We attempted training each model both with and without class weights, noticing a slight improvement in performance without class weights. This observation highlights the significant impact of class weights on model performance, sometimes leading to better results and in other cases, worse.

Following the initial development and assessment of the first three models, further iterations were pursued to address identified limitations and explore alternative variations. Recognizing the constraints of the initial custom model, adjustments were implemented to mitigate underfitting issues. These modifications encompassed fine-tuning parameters such as batch size, the addition of a dense layer, reduction of dropout rates, and extension of early stopping patience. Notably, class weights were removed to refine model performance. With these modifications, we obtained the “model_custom_mod” which yielded much better results than the three initial models.

Expanding our investigation, we introduced two new models, designated as Model A and Model B. Model A synthesized insights from prior iterations, joining aspects from both the “model_custom_mod” and “model_transfer”. Utilizing the VGG16 architecture as its foundation, Model A incorporated L2 regularization

to counter overfitting, maintained balanced dropout rates for optimal model complexity, and utilized RMSProp as an optimizer to enhance performance.

Concurrently, Model B aimed to improve on the general idea of “model_transfer_2layers” by unfreezing the final layers of the VGG16 base model and introducing batch normalization for accelerated training. Key attributes of Model B included a higher neuron count post-base model, a slightly more assertive dropout rate, and the adoption of a smaller learning rate to refine the training process.

After constructing each model, a callback list was assembled to enhance training and monitor performance. This included ModelCheckpoint to save weights at intervals, EarlyStopping to halt training when validation performance stagnates, ReduceLROnPlateau to adjust learning rates, and TensorBoard for some insightful visualizations. Once configured, the model underwent training and was saved for subsequent evaluation.

7. Model Evaluation

In this section, we will visualize metrics and plots for the best model, “model_b”, and in some cases, compare it with the second-best model, “model_custom_mod”.

To initiate the evaluation, we will begin by examining the training curves (Figure 1) and comparing them with those of the “model_custom_mod”.

Upon comparing the two models, it's evident that “model_b” achieved significantly higher validation accuracy than “model_custom_mod”. Focusing specifically on the curves of “model_b”, we observed that it reached the 100-epoch limit which is not problematic because the validation accuracy appeared to plateau. When comparing the validation curve with the training curve, there is arguably some overfitting to the training set, but given that the validation accuracy is not deteriorating, we can conclude that these training curves appear satisfactory.

To provide a comprehensive evaluation beyond just accuracy, key overall performance metrics were calculated to assess “model_b” and compare it with other models (Table 1). The primary metric of interest was the weighted F1 score, where “model_b” achieved the highest score of approximately 0.36.

For a more detailed evaluation, the objective is to gain deeper insights into the model's behavior. Specifically, we aim to examine the F1 score per label (Figure 2) to understand how well the model performs across different categories. Ideally, we want a balanced model where performance is distributed evenly among all classes. As shown in Figure 2, compared to “model_custom_mod”, “model_b” demonstrates a more balanced performance. It's noteworthy that there is a class with a 0 F1 score in both models (Label 77 - 'pilomatricoma'). This label represents one of the least represented classes in the training set, along with 'xanthomas' and 'pustular psoriasis', each having only 37 images. This lack of sufficient training data could be a significant factor contributing to the notable errors observed for this label in both models.

Another interesting aspect to examine is the relationship between the representativeness of each label in the training set and the corresponding F1 score for that label (Figure 3). As observed in the image below, in some cases where the training representativeness is lower, the F1 score also tends to be lower, which explains the model's poorer performance on certain labels. This issue could potentially be addressed by adding more images to the underrepresented labels in the dataset.

To conclude the model evaluation, we created a plot that provides a comprehensive overview of the number of correct and incorrect predictions for each label (Figure 4). This plot offers valuable insights into the model's performance across different categories, highlighting areas of strength and potential improvement.

8. Multi Input Model

Until now, the models we've discussed have relied solely on image inputs. However, in this section, we'll introduce a multi-input model that incorporates both images and additional key features such as "fitzpatrick_scale", "fitzpatrick_centaur", "nine_partition_label", and "three_partition_label". We

implemented this using the Keras Functional API to build the model and created a custom multi-input generator, along with additional preprocessing steps to accommodate these diverse inputs.

This multi-input model integrates the previously best-performing image processing model, "model_b", with frozen layers. It features two distinct input branches: one for image data processing using "model_b" and another for non-image features. The image input is processed by "model_b", flattened, and then combined with the non-image features. The combined features are passed through a sequential-like top model, which includes fully connected layers (the only trainable layers), to perform further processing and classify the data into the specified number of output classes using softmax activation.

After printing some metrics (Table 1), we observed that even without any specific model architecture optimization, the multi-input model achieved a higher weighted F1 score (approximately 0.41) compared to the best-performing image-only model. This outcome was expected, as the multi-input model benefits from accessing more information through discriminative features from both image and non-image inputs.

9. Future Work

The primary focus of this project was on image-only models, and as a result, the exploration of the multi-input model was limited. However, for future work, it would be intriguing to delve deeper into improving the multi-input model by experimenting with different model architectures and implementing more detailed preprocessing steps. Additionally, exploring individual label performance in the image-only models could be insightful to understand why certain labels perform poorly. One suggestion would be to acquire more images for these specific labels and assess if the model's performance improves as a result. This exploration could provide valuable insights into enhancing the overall performance and effectiveness of both model types.

10. Conclusion

In conclusion, this report outlines the development and evaluation of deep learning models tailored for dermatology classification tasks. The project addressed the challenges posed by the visual complexity and variability of skin conditions, utilizing a dataset sourced from the “FITZPATRICK17” repository. Through iterative testing, we addressed initial limitations such as underfitting and explored alternative architectures to enhance performance. Notably, "model_b" emerged as the best performing among the models that relied solely on image inputs, achieving higher validation accuracy and weighted F1 score compared to other image-only models. However, the multi-input model, which incorporates both images and additional key features, and integrated the previously best-performing image processing model, "model_b", with frozen layers, achieved the highest weighted F1 score of approximately 0.41. This outcome underscores the significance of leveraging diverse data sources and integrating additional features for improved classification accuracy.

Moving forward, additional exploration is warranted to optimize the multi-input model further and delve into label-specific performance to enhance overall effectiveness. Overall, this project underscores the potential of deep learning in dermatology diagnostics, paving the way for more accurate and efficient skin condition classification and contributing to the ongoing evolution of medical diagnostics.

11. Bibliography

[1] M. Groh. fitzpatrick17k [Online]. Available: https://github.com/mattgroh/fitzpatrick17k

[2] G. Rohini, “Everything you need to know about VGG16” Sep 2021. [Online]. Available: https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

[3] M. Groh, C. Harris, L. Soenksen, F. Lau, R. Han, A. Kim, A. Koocheck, O. Badri, “Evaluating Deep Neuronal Networks Trained on Clinical Images in Dermatology with the Fitzpatrick 17k Dataset” Apr 2021. [Online]. Available: https://arxiv.org/pdf/2104.09957

Annex

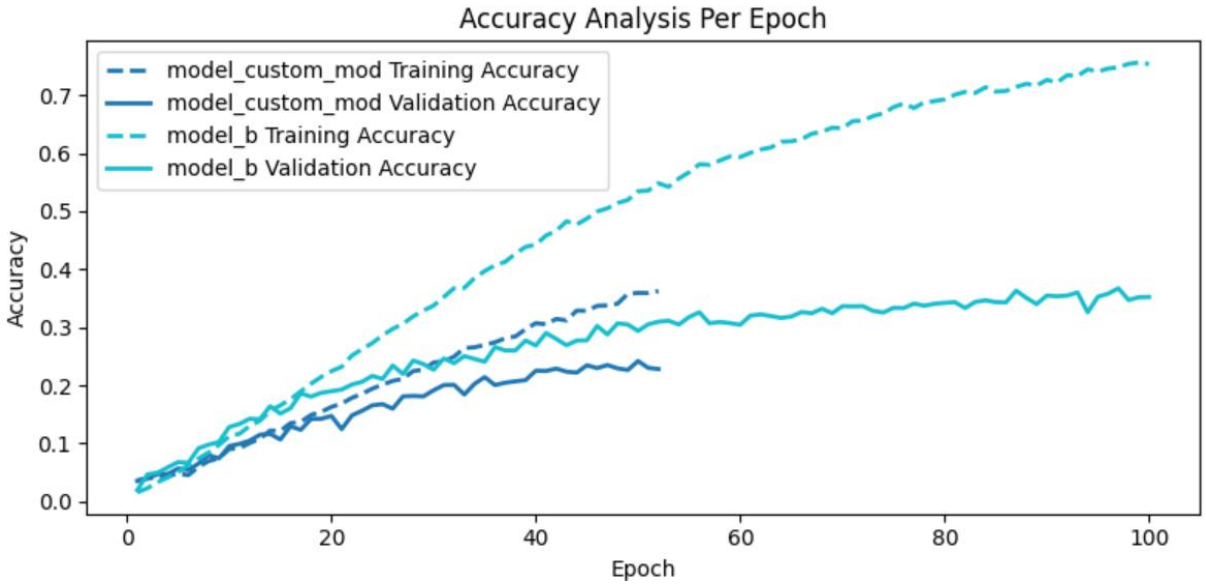

Figure 1 - Training curves (model_custom vs model_b)

Model Name	Micro Avg.			Weighted Avg.			Accuracy
	Precision	Recall	F1-score	Precision	Recall	F1-score	
model custom	0.16	0.1	0.1	0.16	0.16	0.12	0.16
model_custom_wc	0	0.01	0	0	0.01	0	0.01
model transfer	0	0.01	0	0.01	0.06	0.01	0.06
model_transfer_w_cw	0	0.01	0	0	0.01	0	0.01
model_transfer_2layers	0	0.01	0	0	0.04	0	0.04
model_transfer_2_layers_w_cw	0	0.01	0	0	0.01	0	0.01
model_custom_mod	0.22	0.19	0.19	0.23	0.22	0.21	0.22
model_a	0.04	0.05	0.03	0.05	0.1	0.06	0.1
model_b	0.37	0.38	0.36	0.39	0.36	0.36	0.36
multi_input_model	0.51	0.38	0.42	0.47	0.4	0.41	0.4

Table 1 - Performance metrics

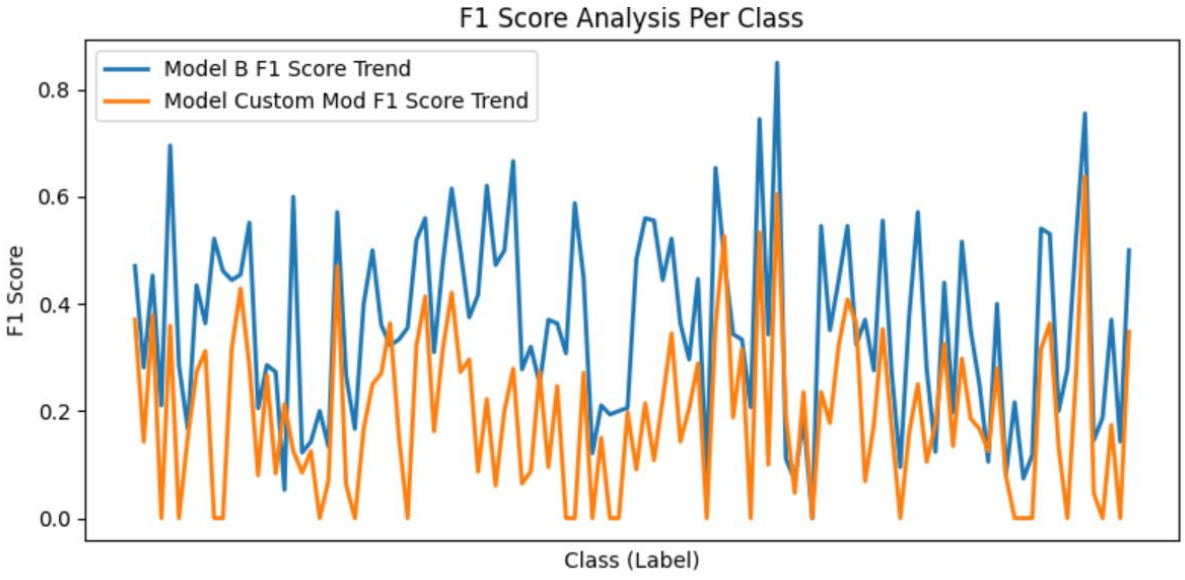

Figure 2 - F1 score analysis per class (model_b and model_custom_mod)

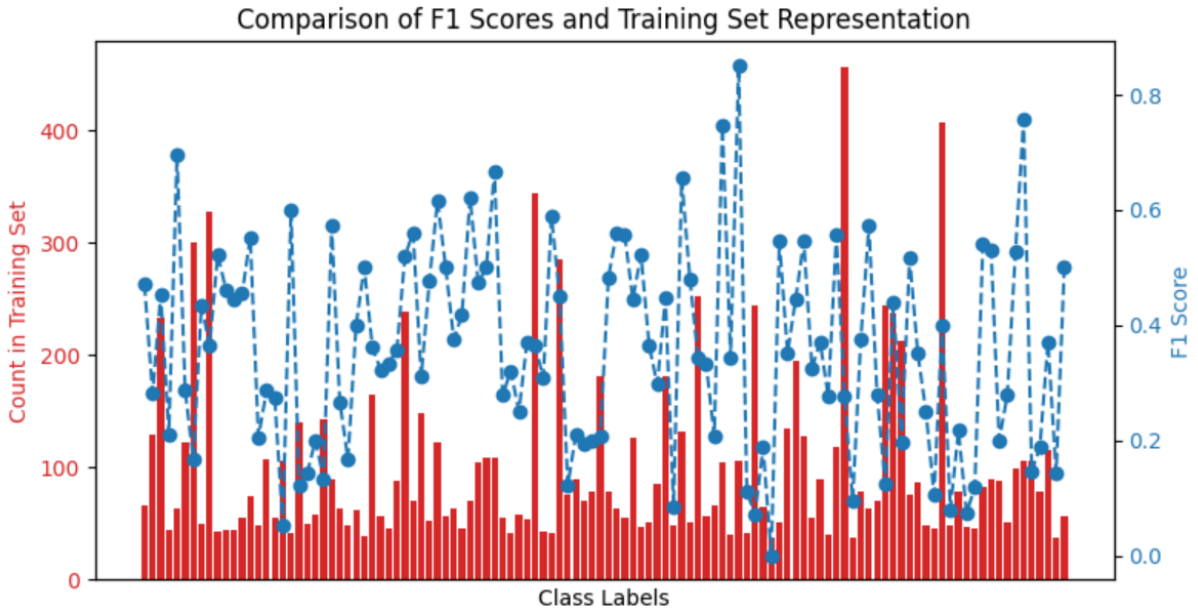

Figure 3 - Comparison of F1 scores and training set representation.

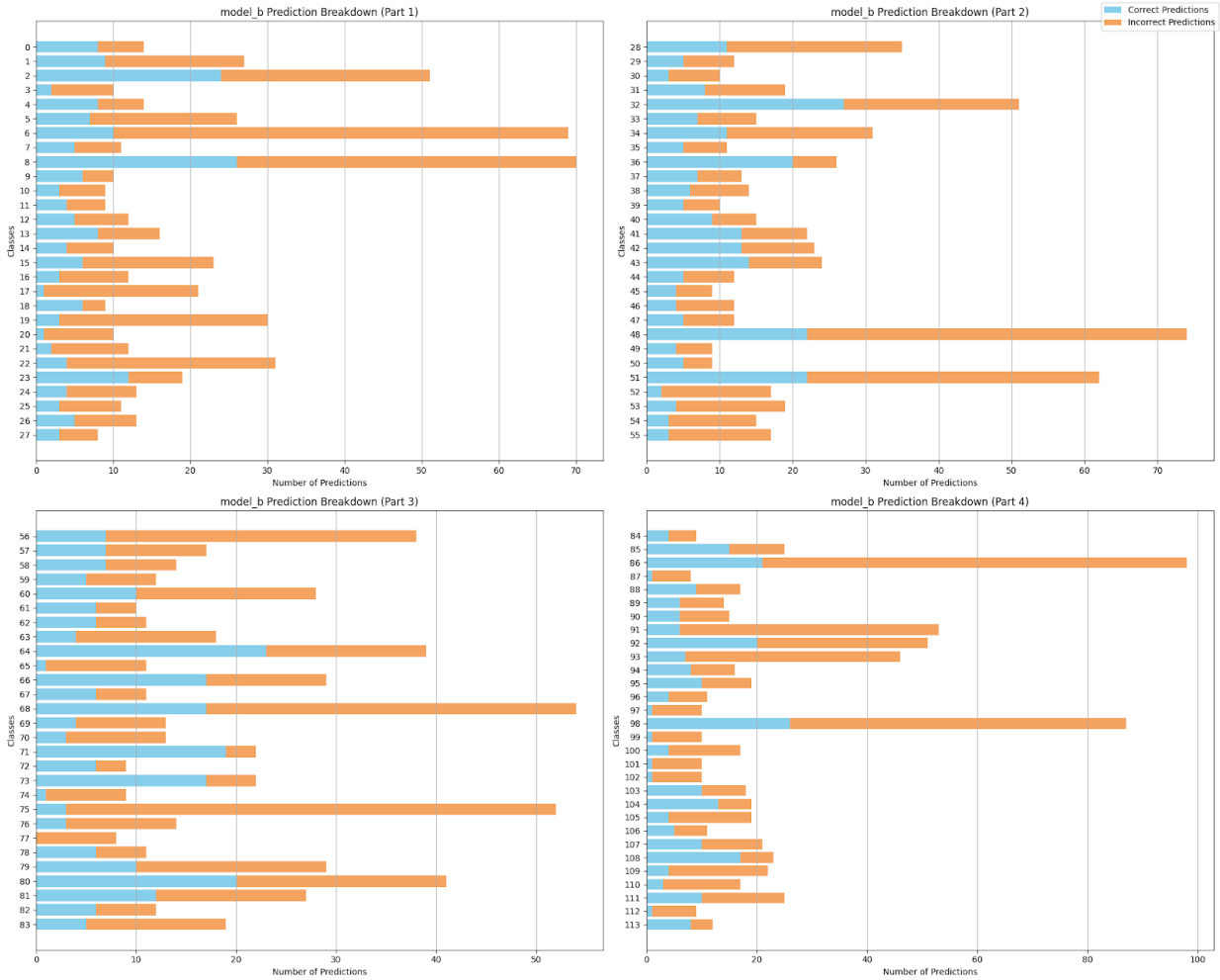

Figure 4 - Class overview (model_b)

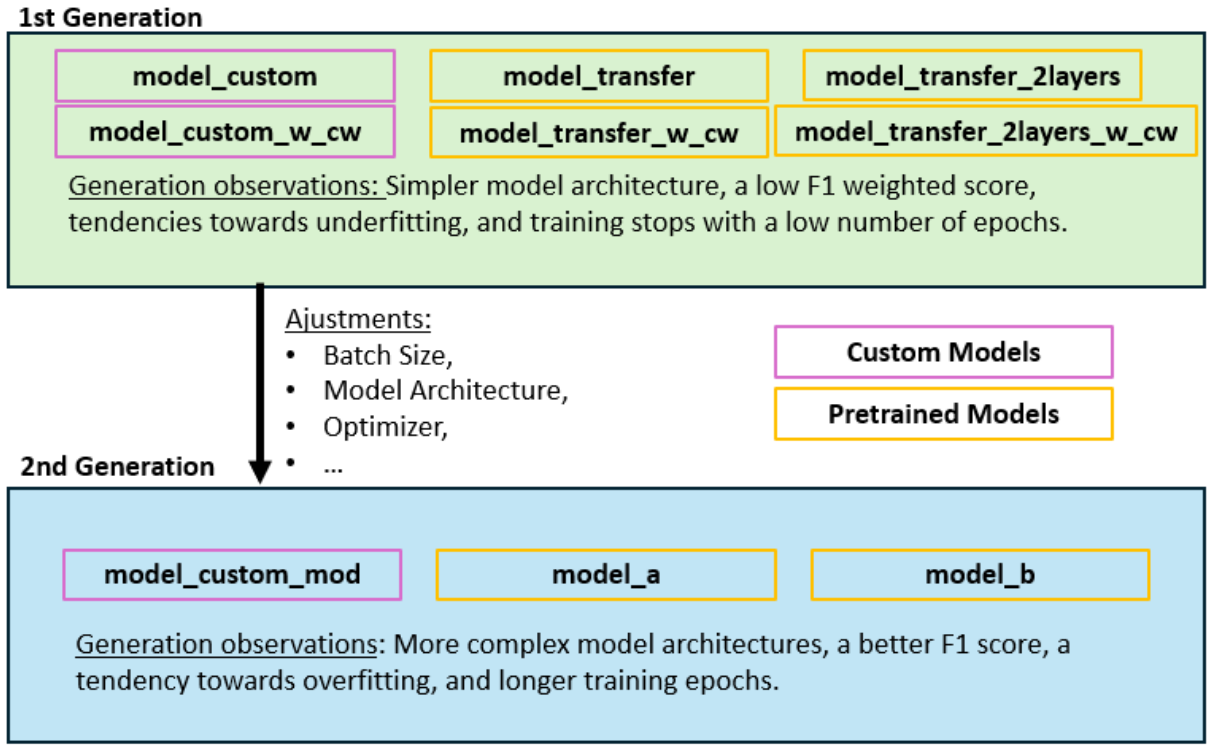

Figure 5 - Methodology schema