Galaxy Classification using Convolutional Neural Networks

Daniel Sobral Blanco[*]
Department of Theoretical Physics,
University of Geneva.

June 7, 2023

Abstract

We present a framework to build, compile and train Convolutional Neural Networks to classify the morphology of galaxies, inspired by the Galaxy Zoo Kaggle challenge of 2014. We split the original Kaggle challenge into two tasks and explore a variety of preprocessing methods and model architectures to find a viable model. Given the nature of the dataset, each task could be solved as a classification or a regression problem. We compare the performance of the most interesting models by looking at the training time and validation metrics.

Disclaimer: The code designed for the project probably surpasses by a lot the requirements of the course. The necessary details to run the code from a zsh terminal are given in the `README.md`. The `.py` scripts include docstrings describing the classes and methods usage. These can be also found in the `documentation.pdf` file. While I do not expect a thorough revision of the code, I would appreciate any (positive or negative) feedback on the overall structure and implementation of the code.

Notes: The results presented in this report shall be reproduced by setting the random seed value to `seed`= 48. The models are saved in the `saved_models` folder, which contains the model `.h5` file, the model diagram and the model history pickle file. Metric plots and confusion matrices are stored in the `metrics_plots` folder. The `image_plots` folder contains the galaxy images included here. The `logs` folder contains `.txt` files with the saved configuration file for each model. Finally, we include a Jupyter notebook `analysis.ipynb` to inspect and test some of the results.

1 Introduction

Understanding the origins of galaxies and their evolution is among the most fundamental questions faced by astronomers and cosmologists. Galaxies, in their myriad shapes, sizes, and colors, provide invaluable clues to the mysteries of the universe. The morphology of galaxies, ranging from enchanting spirals to colossal ellipticals, is believed to be intrinsically linked to the physics governing their formation and evolution.

The collection of vast amounts of images of distant galaxies, facilitated by the relentless advancement of telescopic technologies, has given rise to large datasets of galaxy images. Traditionally, the task of sorting and classifying these images based on their morphology was performed by humans. The Galaxy Zoo project [1], through its successful citizen science initiative, harnessed the power of crowdsourcing to classify galaxy shapes. However, the efficiency of manual classification significantly diminishes as datasets grow to include hundreds of millions or even billions of galaxies.

In the face of this challenge, the Galaxy Zoo Kaggle Challenge [2] of 2014 aimed to develop automated metrics that could reproduce the probability distributions derived from human classifications. The objective was to build an algorithm that behaves as well as the crowd in classifying galaxies into categories based on their images.

This project takes inspiration from this Kaggle challenge and provides a framework to build, compile, and train Convolutional Neural Networks (CNNs) for the purpose of classifying galaxies

[*]Email: daniel.sobralblanco@unige.ch

based on their morphology. The original challenge is divided into two simplified tasks for this project, each presenting the opportunity to be solved as either a classification or regression problem. These tasks are formulated based on the hierarchical structure of the classifications in the original Kaggle challenge. The report that follows will detail the methodologies employed, experiments conducted, results obtained, and a comprehensive discussion and conclusion of the findings.

The complexities inherent in this problem, such as the hierarchical nature of galaxy classifications and the format of the labels, which are floating point numbers representing the fraction of participants assigning a certain classification, present unique challenges and opportunities. Through this project, we aim to explore a variety of pre-processing methods, model architectures, and approaches to design a suitable model for this intriguing problem. The performance of the models is evaluated based on their training time and validation metrics, providing insights into their efficacy in handling this demanding classification task.

While venturing through the intricate cosmos of data science and machine learning, one seldom has to travel alone. The assistance of AI has become a precious resource for researchers, providing a helping hand in routine tasks as well as the most complex problem-solving scenarios. In this project, we have incorporated the help of ChatGPT, a powerful language model developed by OpenAI. This integration, though not originally a part of the project's purpose, represents an exciting self-imposed challenge to explore the untapped potential of ChatGPT. Despite the fact that this move diverges from the original project requirements, it provides an opportunity to push boundaries and see what this AI is truly capable of. The purpose is twofold: to explore its ability in understanding and generating human-like text in an effort to better articulate complex project elements efficiently, as well as to improve our collaborative workflow and streamline the documentation process. By weaving ChatGPT into the fabric of our project, we seek not only to accomplish our galaxy classification goals but also to highlight the impressive potential of AI in amplifying human capacity in scientific research and exploration.

2 Methodology

2.1 The Galaxy Zoo dataset

Our data for this project is derived from the Galaxy Zoo Kaggle challenge dataset. This dataset consists of JPG color images of galaxies, each of 424x424 pixels, along with corresponding labels assigned based on the probabilities derived from human classification efforts. Hence, the labels are represented as floating-point numbers rather than binary integers.

Given computational limitations, we resized the images to a more manageable dimension of 64x64 pixels. To mitigate the inevitable loss of resolution associated with this downscaling, we could first crop the images to extract their central portion, where the galaxy and thus the relevant features typically lie. The resulting image size of 64x64 is a standard choice in similar machine learning tasks, balancing computational feasibility with minimal loss of critical classification features.

To prepare the data for our machine learning tasks, we developed custom pre-processing functions contained in the utils.py script. The image pre-processing functions incorporate procedures for cropping, resizing, and normalizing the pixel values of the images. Despite the option to convert the images to grayscale, we decided to proceed with the original RGB format for training our models.

Regarding the labels, we have included an option to load them in a one-hot encoded format, making them more suitable for a classification task. For each task, we select the relevant labels, and we offer the flexibility to choose the minimum probability value considered for each task. Specifically for task 1, we proposed using a threshold probability of 0.8, which means discarding all images where any probability label falls below 0.8. For the second task, we use the full image dataset, hence setting the minimum probability to 0.0. This approach aims to ensure a higher degree of certainty in the label classifications for our training data.

Finally, we divide the dataset into training, validation, and test sets. This splitting allows us to train our models on one subset of the data and evaluate their performance on a separate subset, helping to ensure the models' ability to generalize to new data.

In the following sections, we will discuss the specific architecture of the Convolutional Neural Networks (CNNs) utilized, and the details of the training process.

2.2 Model Architecture

Given the nature of the tasks, we designed a Python class capable of building a variety of Convolutional Neural Network (CNN) architectures, as the suitable architecture can vary depending on the task.

For the first task, we found that a particular type of architecture tends to perform better:

$$\text{Input} + n \cdot (\text{Conv2D} + \text{MaxPool2D}) + \text{Flattening} + n \cdot (\text{Dense} + \text{Maxout} + \text{Dropout}) + \text{Output} \quad (1)$$

The class we designed allows for creating this architecture with a flexible number of convolutional blocks and dense blocks. We shall call this type of model architecture as the ***base*** models. The class also allows for the possibility to implement *batch normalization* in each convolutional block. For the flattening layer, we considered both standard Flatten and GlobalMaxPooling2D strategies. However, while *GlobalMaxPooling2D* increased training time, it did not appear to improve performance. As a result, we opted to only use the standard Flatten layer.

The second task is more challenging due to an increased number of classes and a hierarchical structure within these classes. In particular, the predictions for Class 2 and Class 7 depend on the predicted subclass of Class 1, as determined in the first task. Let's write down this decission tree:

Class1: *Is the galaxy simply smooth or rounded, or has it some feature like a disk?*

- Class 1.1 = SMOOTH.

  Class 7: *How rounded is it?*

  - Class 7.1: ROUND
  - Class 7.2: ELIPSE
  - Class 7.3: CIGAR

- Class 1.2 = DISK

  Class 2: *Can it be a disk viewed edge-on?*

  - Class 2.1: YES
  - Class 2.2: NO

- Class 1.3 = STAR/ARTIFACT

Our model needs to account for this hierarchy. To that end, we designed our class to construct two types of models for this task:

- ***Hierarchy* models:** We modified the architecture used in task 1 to become a multi-output setup. This modification enables us to enforce the hierarchical relation between the outputs using custom Lambda layers. Specifically, these layers weight the probabilities of the outputs for Class 2 and Class 7 with the predicted probability of Class 1.

- ***Branch* models:** For these models, we used the task 1 model architecture to create a model with three identical branches. Each branch receives the same image as input, processing the images in independent convolutional layers. We then take the outputs from the Class 1 branch and concatenate them with the Class 2 and Class 7 branches respectively, right after the Flattening layer at the end of the convolutional blocks. To enforce the hierarchy of the outputs for Class 2 and Class 7, we use the same Lambda operations as in the hierarchical models.

The specific parameters chosen for the models utilized in each case will be further discussed in the Results section, providing a detailed insight into the decision-making process and the performance trade-offs made. Additionally, we have found some architectures to be more successful than others in performing these tasks. The architecture diagrams of the most successful models for each case will be included in an Appendix for a comprehensive view of the structural layout of the networks, which contributed significantly to their performance.

These designs aim to leverage the inherent structure of our classification tasks and ensure our models adequately reflect this structure for optimal prediction performance. In the following section, we discuss how these models were trained.

2.3 Training

In this project, the approach to model training varied depending on whether we viewed the tasks as classification or regression problems. It's crucial to note that both perspectives are valid and can be applied by appropriately encoding the labels in one-hot format or not. In fact, task 2 can also be considered under both approaches. This involves converting the maximum values of the subclasses in Class 2 and Class 7 to 0 or 1, conditional on the parent Class 1 subclass.

The dataset was partitioned in a two-step process. Firstly, the loaded dataset was split into training and validation sets by adjusting the training_size parameter. Subsequently, if a distinct test set was desired, a value was set for the test_size parameter, which resulted in the extraction of a test set from the training set. Our strategy was to balance these parameters to achieve roughly equal numbers of samples in the validation and test sets, but leaving the majority of samples for training.

The Adam optimizer was employed universally as our choice for the optimization algorithm. As the state-of-the-art optimization algorithm for tasks of this kind, it has been highly effective, not presenting any limitations or issues that would necessitate exploration of other options.

The choice of loss function depended on the architecture used and the problem type. In general, we used categorical cross-entropy for the classification tasks and mean squared error for the regression tasks. We also considered the focal loss as a potential alternative method to address data imbalance without explicitly using class weights.

Various regularization techniques were explored in this project. Our CNN builder class allows for the implementation of batch normalization in convolutional blocks, and maxout and dropout layers in dense blocks. Although other techniques were considered, these three have proven to be the most effective for the tasks at hand.

We employed data augmentation and class weights as techniques to balance our dataset. Data augmentation was particularly critical in preventing overfitting and enhancing the generalization ability of the models across both tasks. Meanwhile, class weights were not essential when removing samples with labels below a 0.8 probability but became highly significant when the entire dataset was considered. This was due to the presence of class imbalance, which affected model performance. The implementation of class weights posed an interesting coding challenge, especially in the multi-output scenarios, as the standard Keras pipeline was not fully suitable. Therefore, we devised custom loss functions that manually weighted the standard Keras loss functions.

2.4 Incorporation of ChatGPT-4 in Project Workflow

This project not only aimed to tackle the Kaggle Galaxy Zoo challenge, but also sought to explore the potential of AI language models, specifically OpenAI's ChatGPT-4, in project development and coding assistance.

1. **General Purpose Project Assistance:** In the initial phase of the project, ChatGPT functioned as a general-purpose project assistant. By feeding the assistant with the project details and conceptual ideas, it consistently provided valuable insights and suggestions. A striking instance of this was when ChatGPT proposed the use of the focal loss function as an alternative to conventional class weights. This advice proved instrumental, as it significantly improved the model's handling of class imbalances.

2. **Coding Assistance:** The role of ChatGPT advanced to a coding assistant as the project evolved. By providing detailed prompts, the assistant generated Python code tailored to meet the specific needs of the project. Some sections of the codebase, specifically the custom losses and custom layers scripts, were entirely generated and debugged with the help of ChatGPT. It was also responsible for crafting the docstrings throughout the code, providing clear and concise explanations of the functionality. Although the assistant does not directly interpret or debug code, it was a valuable tool for understanding Python error messages and proposing solutions.

3. **Writing Assistance:** A significant portion of this document was generated by ChatGPT, acting as a writing assistant. By providing the assistant with a general overview of the project and some directives on the structure of the report, it was able to generate entire sections and subsections, together with the desired LaTeXformat. The eloquence of the language used by the assistant often surpassed that of the user's, providing a level of polish to the final report. It's worth noting, with a hint of humour, that any grammatical or typographical errors found within this document can be attributed to the assistant and not the user.

Benefits and Drawbacks: The most significant advantage of utilizing ChatGPT was the speed and ease of code development, especially when dealing with new or unfamiliar coding tasks. The assistant effectively negates the need to manually delve into extensive code documentation, instead offering common solutions that save hours of research. However, the cutoff of training data for ChatGPT, which only goes up until August 2021, is a noteworthy limitation. This means that any methods deprecated or developed after this date might not be recognized or correctly handled by the assistant. Nevertheless, it always acknowledges this limitation and strives to provide the best possible assistance, albeit requiring more careful testing and scrutiny by the user.

Notable Contributions: Specific examples where ChatGPT made a significant contribution include the suggestion to use focal loss and the generation of entire scripts. More instances will be discussed in the Results section. Overall, the usage of ChatGPT undoubtedly enhanced the speed, efficiency, and quality of the project development.

3 Experiments and Results

Let's explore the dataset in more detail. As mentioned in Section X, the dataset consists in JPG, RGB images with a size of 424×424 pixels. They come together with a .csv file that contains the classification labels. This classification labels are float numbers, normalized between 0 and 1, that represent the probabilities for a galaxy to belong to a particular class. These probabilities have been derived previously from human classification efforts. We show the first five of them in Table 1 for ilustration.

3.1 Data exploration and agumentations

We discuss and test our preprocessing and data augmentation methods. The nature of the problem changes from one task to the other, but we have allowed for flexibility on the preprocessing functions to be able to tackle each of the tasks in the optimal way.

Generally speaking, the first task can be solved as a classification problem, in which we can use one-hot encoded labels of the `Class1` columns in Table 1. For this task, we are required to select the galaxies with at least 0.8 probability to belong to any of the `Class1` subclasses. This reduces the effective dataset to approximately 25,000 galaxies. In principle, this subset of the data contains galaxies that present sharper features and are easier to distinguish.

On the other hand, for the second task, we have to include the `Class2` and `Class7` columns. This adds another dimension to the problem by introducing the inherent hierarchy of the Galaxy Zoo Challenge decision tree: the new classes are subordinated to `Class1`. In particular, a galaxy can only belong to `Class2` if it belongs to `Class1.2` in the first place. Similarly, a galaxy can only belong to `Class7` if it belongs to `Class1.1`. We have found that for this case, where we have to adapt the model architecture to enforce the decision hierarchy, it is generally more effective to see the task as a regression problem, directly using the normalized probability labels.

GalaxyID	Class1.1	Class1.2	Class1.3	Class2.1	Class2.2	Class7.1	Class7.2	Class7.3
100008	0.383147	0.616853	0.000000	0.000000	0.616853	0.201463	0.181684	0.000000
100023	0.327001	0.663777	0.009222	0.031178	0.632599	0.000000	0.135082	0.191919
100053	0.765717	0.177352	0.056931	0.000000	0.177352	0.000000	0.741864	0.023853
100078	0.693377	0.238564	0.068059	0.000000	0.238564	0.408599	0.284778	0.000000
100090	0.933839	0.000000	0.066161	0.000000	0.000000	0.494587	0.439252	0.000000

Table 1: Normalized probabilities for each galaxy class. The 'GalaxyID' is the galaxy tag, which matches with the names for the galaxy images.

3.1.1 First task

We prepare the `Class1` columns labels in one-hot encoded format using a dedicated function, selecting only those samples that belong to one of the subclasses with a probability of at least 0.8. This reduces the dataset to a subset of 24,275 galaxies. The distribution of galaxies among the `Class1` subclasses is as follows:

```
Counts of Class1 Subclasses:
Class1.1 (Smooth): 8,132 samples
Class1.2 (Features/Disk): 16,141 samples
Class1.3 (Star/Artifact): 2 samples
```

Hence, a significant class imbalance is observed. The standard approach for training in such cases is to apply class weights, downweighting the majority classes to give more importance to the minority classes. This helps the model focus on the more challenging cases and improve generalization. However, it is worth noting that the model we have designed is deep enough to distinguish between the two majority classes effectively (which are also imbalanced between them), while the `Class1.3` samples do not pose a problem. This is because the two samples have been included in the training set during the splitting into training, validation, and test sets.

In terms of image preprocessing, resizing the images to 64×64 pixels appears to be sufficient for this task. This may be due to the fact that the images are more distinguishable, at least from a human perspective. Consequently, the loss of resolution resulting from downsizing the images does not significantly impact the model's ability to learn the relevant image features for this task (see Fig. 1).

3.1.2 Second task

Given the hierarchical nature of galaxy classifications, the second task presents a different challenge. Initially, one could approach this task as a classification problem by converting the labels to a one-hot encoded format. We designed a function that accomplishes this by transforming the maximum values of the `Class2` and `Class7` columns based on the values in `Class1.1` and `Class1.2`. However, we found that models struggle to learn effectively when provided with one-hot encoded labels for this task. Therefore, we solve this task effectively as a regression problem.

To simplify the task, we have chosen a threshold probability of 0.5. Galaxies with maximum probabilities below this threshold are extremely difficult to classify, even for humans, and would pose a similar challenge for a machine learning model. This choice only slightly reduces the dataset, leaving us with 60,023 samples. These galaxies are classified as follows:

```
Counts of Class1 Subclasses:
Class1.1 (Smooth): 25,874 samples
Class1.2 (Features/Disk): 34,105 samples
Class1.3 (Star/Artifact): 44 samples

Counts of Class2 Subclasses among those assigned to Class1.2:
Class2.1 (Normal Disk): 6,626 samples
```

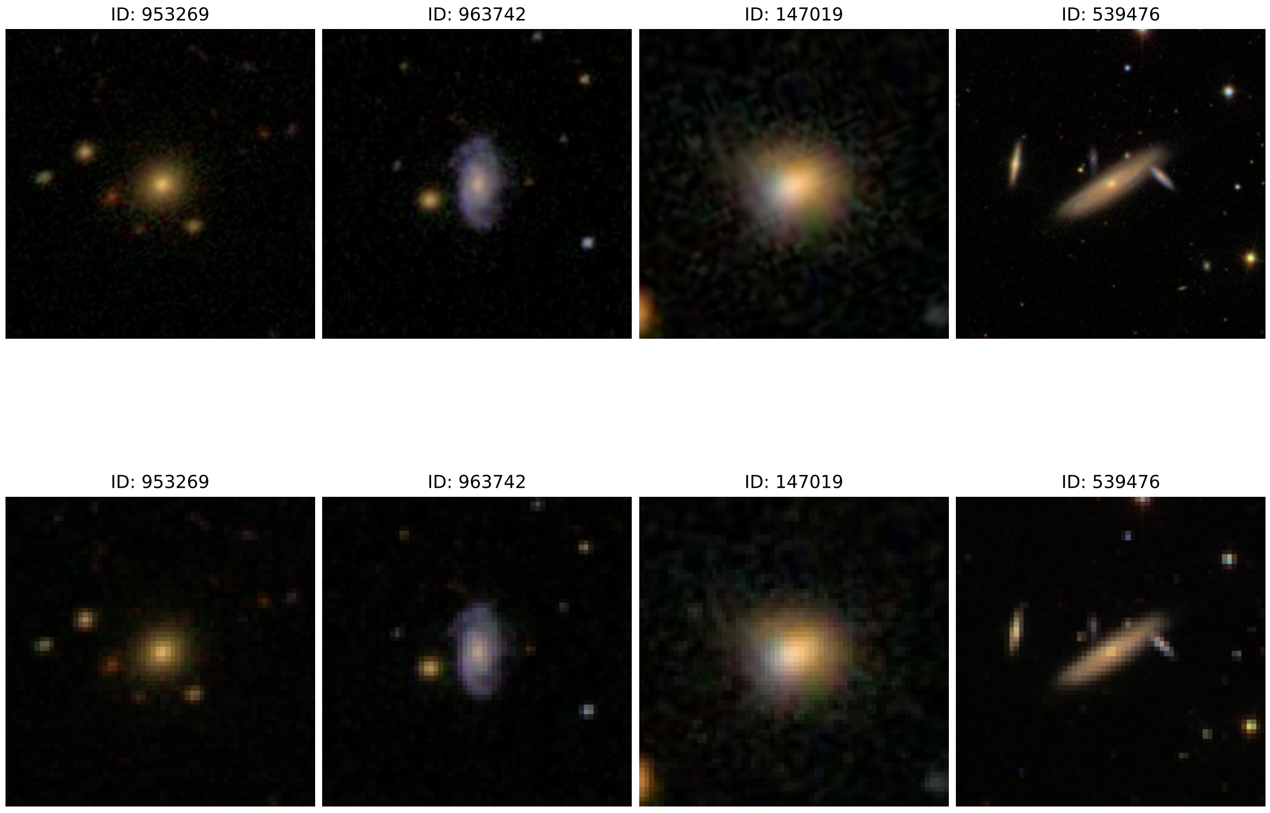

Figure 1: Randomly selected samples among the relevant images used for the first task. Plot (a) are the images as they come in the dataset. In (b) are the same images but downscaled to 64×64 pixels. The loss of resolution is evident, but the features are still recognizable.

```
Class2.2 (Edge-on viewed Disk): 27,479 samples

Counts of Class7 Subclasses among those assigned to Class1.1:
Class7.1 (Round): 11,733 samples
Class7.2 (Elliptical): 12,628 samples
Class7.3 (Cigar): 1,513 samples
```

We observe significant class imbalance across all the classes, which is not trivial in this case. Although the `Class1.3` objects are not actually galaxies, and the other subclasses do not have any samples from this parent subclass, removing them would improve the models' capability to classify objects into the `Class1` subclasses. However, we decided not to remove any samples in any case, as it would be considered cheating. To address the class imbalance issue, we explored two approaches:

- We introduced a class weight calculator that assigns a weight to each class, computed as the logarithm of the inverse frequency of appearance for each subclass. Taking the logarithm is optional, but we used it to prevent excessively large weights for the underrepresented classes, mitigating any negative effects that could arise from the model focusing too heavily on the minority classes. The goal was to improve the overall performance of the model without sacrificing performance on the majority classes. However, this approach proved to be problematic in the multi-output models we designed for this task.
- As an alternative to tackle the class imbalance problem, we explored the use of the *focal loss*. This loss function is specifically designed to address class imbalance in object detection and classification problems (for more details, see Appendix 4). However, similar to the standard class

weights, we had to tune the hyperparameters of the focal loss function to avoid compromising the overall model performance. While it was possible to choose different hyperparameters for each class independently using the computed class weights, we found that using the same parameters for all classes provided the best performance.

Regarding image preprocessing, the second task is more challenging and includes samples with features that are more difficult to learn. Resizing the images to 64×64 pixels may result in a significant loss of information. Therefore, for the second task, we first cropped the center of the original images, extracting a central square of size 256×256 pixels. We then applied the resizing operation to this cropped image. This approach was chosen because the relevant features for classification are typically located in the center of the images, where the galaxies or artifacts are positioned. The choice of 256 for the cropped size is also reasonable, as it is evenly divisible by 64.

3.1.3 Data augmentations

Data augmentation is a commonly used technique in machine learning, especially in tasks related to computer vision. It is employed to artificially increase the size of the training dataset, reduce overfitting, and improve the performance of machine learning models. This is achieved by applying a series of random transformations to the original images, such as rotations, translations, zooming, flips, and sometimes more complex operations like brightness and contrast adjustments.

The primary advantage of data augmentation lies in its ability to create a more robust model by exposing it to a wider range of variations in the data. By doing so, the model is less likely to overfit to specific features of the training data and can generalize better to new, unseen data. Additionally, data augmentation can be particularly valuable in situations where the amount of available data is limited.

However, caution must be exercised when applying data augmentations, as not all types of transformations may be suitable for all kinds of data. For instance, flipping an image may not be a suitable augmentation for certain tasks, as it could alter the meaning of the image. Moreover, excessive augmentation can lead to a model learning features that are not useful or even misleading, potentially harming the model's performance. Hence, the selection of appropriate augmentation techniques is a critical step that should be tailored to the specific task and dataset.

In our experiments, data augmentation proved to be a crucial technique that allowed us to train models with higher learning rates without encountering overfitting issues. For the first task, data augmentation alone was sufficient to mitigate overfitting. We used the following operations for augmentation:

```
data_augmentation_params:
  rotation_range: 90
  width_shift_range: 0.075
  height_shift_range: 0.075
  horizontal_flip: true
  vertical_flip: true
  shear_range: 0.2
  zoom_range: 0.25
```

As shown in Fig. 2, these transformations appear reasonable in the context of galaxy morphological classifications. They introduce more diversity into the dataset with realistic variations while preserving the shapes and features of the galaxies.

However, these transformations proved to be too aggressive for the second task. In this case, we included all galaxies that belong to one of the classes with a probability larger than 0.5, thus increasing the sample diversity. However, this choice also includes samples that are difficult to classify, even for humans, which poses a challenge for any convolutional network. Although data augmentations are still a key tool to prevent overfitting in our second task models, we need to be more cautious with the transformations we introduce. Therefore, we reduced the intensity of the potentially confusing transformations:

```
data_augmentation_params:
  rotation_range: 90
  width_shift_range: 0.01
  height_shift_range: 0.01
  horizontal_flip: true
  vertical_flip: true
  shear_range: 0.015
  zoom_range: 0.15
  fill_mode: nearest
```

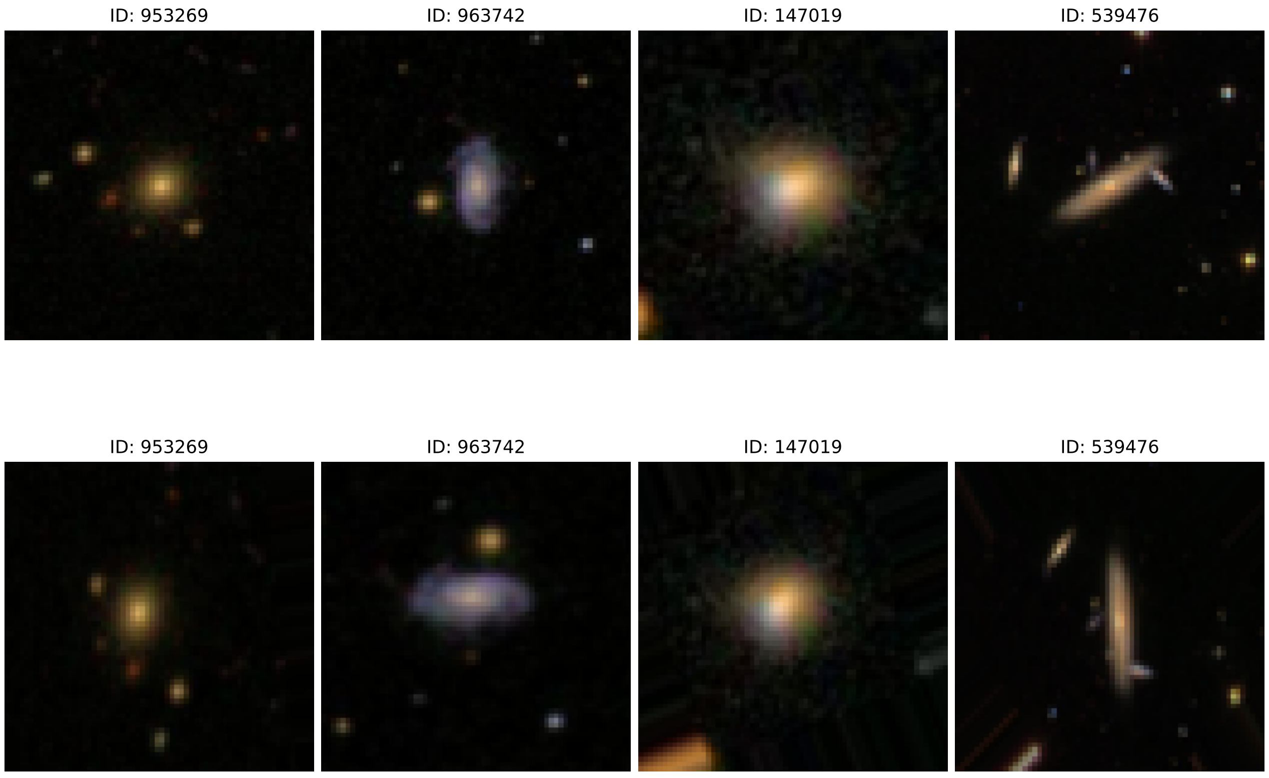

Figure 2: We compare randomly selected downscaled samples from the relevant images used for the first task before (above) and after data augmentation (below). The augmented images were generated using our custom class, `MultiOutputDataGenerator`..

3.2 First Task: classification into Class 1 subcategories.

The first task involves building a convolutional neural network (CNN) to classify the morphology of galaxies into the three primary categories of the Kaggle challenge decision tree. The Galaxy Zoo project released the images to the public and posed the question: "Is the galaxy simply smooth or rounded, or does it have some features like a disk?" The `Class1` subclasses correspond to the three possible answers to this question:

- `Class1.1`: Smooth.
- `Class1.2`: Features or disk.
- `Class1.3`: Not a galaxy, but a star or artifact.

We constructed a CNN to classify the galaxies into these three classes. As mentioned earlier, we only used samples with probabilities above 0.8, which reduced the effective number of samples to 24,275 galaxies. We conducted several experiments by changing the model architecture and regularization techniques to find the best performing model for this task, evaluated using the accuracy and F1-score metrics. Additionally, we computed the confusion matrices to gain better insight into the performance for each subclass.

The architecture of the best performing model is as follows:

- **Convolutional Blocks:** We utilized three `Conv2D` layers with 32, 64, and 128 neurons respectively, each with a filter size of (3,3) and `ReLU` activations. After each `Conv2D` layer, we applied `MaxPooling2D` layers with a pool size of (2,2) to reduce dimensionality. The extracted features were then passed through a `Flatten` layer.
- **Dense Blocks:** We included two `Dense` layers with 512 and 256 neurons respectively, each with `ReLU` activations, followed by `Maxout` layers. The `Maxout` layers group and average the output, effectively reducing the number of neurons by a factor of 1/2. This averaging helps smooth out large fluctuations in the output values of the `Dense` layers. Additionally, we incorporated `Dropout` layers after each `Dense` layer for regularization, using a dropout rate of 0.25.
- **Output:** The final `Dense` layer consists of 3 neurons (corresponding to the number of classes) with a `softmax` activation function.

We trained this architecture using the categorical cross-entropy loss, which is suitable for multilabel classification problems. The Adam optimizer, a widely used tool in convolutional networks, was chosen. To address overfitting, in addition to dropout regularization and data augmentation, we implemented `EarlyStopping` with a patience of 15, saving the model weights of the best epoch.

The model training process took 97 epochs and approximately 53 minutes, achieving an accuracy of 98% on the selected subset of samples. Evaluating the model on a testing set extracted from the training samples yielded the following results:

```
----- EVALUATING MODEL -----
Evaluating the model on the testing set
122/122 [==============================] - 3s 22ms/step

Test loss: 0.04486474022269249
Test accuracy: 0.9840371012687683
Test F1_score: 0.9841188788414001
```

The similar values for accuracy and F1-score indicate that the model is performing well and can effectively distinguish between classes (see Figs. 3 and 4). However, since there are no samples corresponding to `Class1.3` in the test set (the two existing samples were included in the training set), we cannot expect the model to effectively classify this category. In fact, when training the model with the full dataset, the same architecture, accuracy, and F1-score drop to approximately 85%, even when weighting the classes to give more emphasis to the difficult samples.

Remarks and Observations:

During our experiments, we made several observations:

- Models of this type have a large number of trainable parameters, making them prone to overfitting. However, by employing data augmentation and `EarlyStopping`, we were able to address this issue and obtain a well-performing model. The introduction of additional regularization techniques, such as dropout, led to only a slight improvement in performance without significantly affecting the training time. Therefore, we did not explore other regularization techniques.
- We attempted to add Batch Normalization to the convolutional layers, a method commonly used to expedite and stabilize the training of neural networks by normalizing the layer inputs through re-centering and re-scaling. Surprisingly, in our model, adding batch normalization to

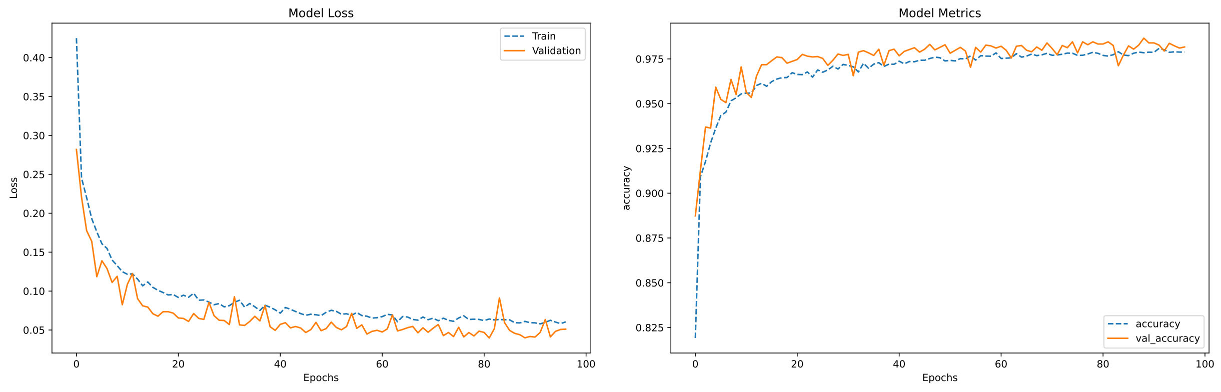

Figure 3: Loss and accuracy as a function of the epochs for the best performing model in the First task. The trends clearly indicate that the model effectively learns to classify the galaxies without any signs of overfitting.

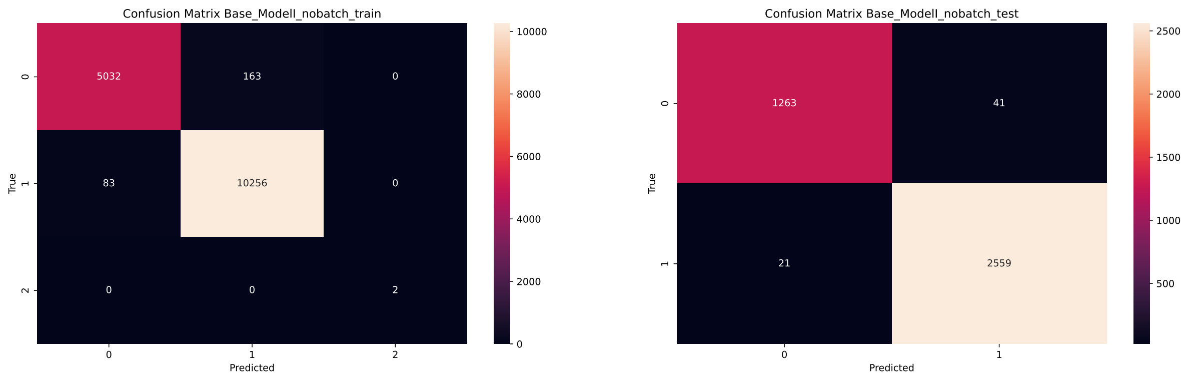

Figure 4: Confusion matrices for the training (left) and testing (right) sets of the best performing model in the First task. Only a few samples of 'Class1.1' (0) and 'Class1.2' (1) are misclassified, which explains the high and similar values for accuracy and F1-score.

the convolutional layers resulted in a less stable training history, with large fluctuations in the loss and metric curves. However, it reduced the training time by approximately 7 minutes while still yielding a model with similar performance on the test set.

- We trained the model without applying any class weighting, and yet it was able to effectively distinguish between classes. We also tested our custom loss function based on the focal loss with moderate hyperparameters on this architecture, but no significant improvement was observed. The performance on the test set was slightly lower but still around 98%. The main advantage of using the focal loss was a reduction in training time by approximately 12 minutes.
- More aggressive class weighting techniques, such as downweighting the majority classes and upweighting the minority classes using the inverse frequency of each class within the sample, seemed to confuse the model, resulting in some More misclassifications even within the majority classes.

3.3 Second Task: hierarchical classification problem.

The Second task adds an extra layer of complexity to the problem. Following the Galaxy Zoo project decission tree, we have to consider now two extra classes which are actually subordinated to the `Class1` subclasses. In summary:

- If a galaxy belongs to `Class1.1`, then it shall also be classified into one of the `Class7` subclasses.

- If the galaxy belongs to `Class1.2`, then it can belong to one of the `Class2` subclasses.

Hence, the new challenge is to build a model which learns in some way about this hierarchy. Numerically, the hierarchy implies that the sum of the probabilities of the `Class2` and `Class7` subclasses must match the value of their respective parent class (see Table1).

First, we tested how well our first task model performed for this task, without any other change to its architecture beyond adapting the output layer to have the number of neurons we need (8 classes, hence 8 neurons). In this case, we cannot regard the problem anymore as a classification problem with one-hot encoded labels, but rather as a regression problem where we aim to predict with high precission a vector of values between 0 and 1. This implies we must change to `sigmoid` activation in the output layers. A regression problem also requires to use mean squared error as the loss function. If we want to implement some class weighting, we have found that the focal loss is the more convenient approach. Finally, meaningful metrics to monitor the training process for this case are MSE and accuracy.

However, we still may want to analyse the performance of the model by looking at how well it did actually classify the galaxies. In that case, we have to transform the predicted probabilities to label classes and use the latter to calculate separately the confusion matrix for each class.

Other approaches can be devised to solve this problem. On the first hand, we can take a ‘sequential’ training approach, where we train a model to classify the galaxies into `Class1` subclasses,and then use the predictions to train other two models, one designed to classify the samples into `Class2` and the other to classify them into `Class7` subclasses. This approach is flexible, in the sense we can filter the relevant samples for each task and optimize the weights for each case separately. It is, however, very computationally expensive and requires a lot of post-processing.

Alternatively, we choose to adapt the architecture designs to make the models learn about the hierarchy between the classes. We took the **base** model architecture designed for the First task and adapted it in two different ways, giving rise to two new families of models. We next analyse the results obtained for the best performing models of each new type. They are designed in such a way that they would also allow us to take the task as a classification problem. However, we have observed that this approach is still suboptimal with respect to the regression approach.

3.3.1 Hierachy models

We called the first new family of models we found to useful as the **hierarhcy** models. As described in Section 2.2, we take the **base** model architecture and modify its structure using `Lambda` and `Multiply` keras layers to enforce the hierarchy between the classes probabilities. In order to do this, we modify the output layer to give three different outputs, one per class. We then take the `Class2` and `Class7` outputs with the correspondent prediction for `Class1` (see Fig.5). This enforces the numerical hierarchy between the probabilities.

The coding challenge here has been that it is not straighforward to implement weights, losses and data augmentations in multi-output set ups. Hence, we had to design custom losses and a custom augmentations generator, based on the standard Keras methods. The details on how they work are contained in the code documentation. Moreover, the preprocessing now also requires to split the labels array into three pieces, and feeding it to the Keras API as a list of arrays.

Sumarizing the results, the best perfoming model of this type uses the same convolutional blocks architecture with the mentioned modifications to the output layers. In this case, we observed that Maxout layers are not really useful, so we have removed them. The Dense block consists in two Layers of 256, 128 neurons followed of Dropout layers with rate 0.5. Data augmentation is still the crucial tool to avoid overfitting, but we reduced the intensity of the transformations (see Section 3.1.3.). To alleviate the issue with class imbalance, which is latent in all the three classes, we introduce focal loss function with $\alpha = 0.2$ and $\gamma = 1.5$. This choice upweights the minority class significantly but does not penalize too much the prediciton on the majority classes. We then train the model with an `EarlyStopping` callback with patience 10, which lets the model train for 37 epochs ($\sim$ 1h 10 min). Evaluating the model over the testing set, we get:

```
----- EVALUATING MODEL -----
```

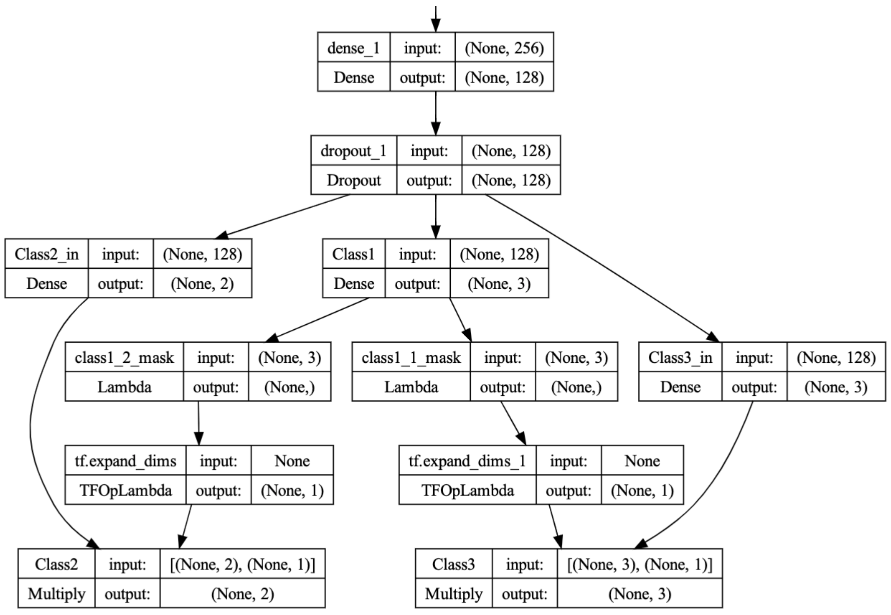
dense_1 | input: | (None, 256)
Dense | output: | (None, 128)
dropout_1 | input: | (None, 128)
Dropout | output: | (None, 128)
Class2_in | input: | (None, 128)
Dense | output: | (None, 2)
Class1 | input: | (None, 128)
Dense | output: | (None, 3)
class1_2_mask | input: | (None, 3)
Lambda | output: | (None,)
class1_1_mask | input: | (None, 3)
Lambda | output: | (None,)
Class3_in | input: | (None, 128)
Dense | output: | (None, 3)
tf.expand_dims | input: | None
TFOpLambda | output: | (None, 1)
tf.expand_dims_1 | input: | None
TFOpLambda | output: | (None, 1)
Class2 | input: | [(None, 2), (None, 1)]
Multiply | output: | (None, 2)
Class3 | input: | [(None, 3), (None, 1)]
Multiply | output: | (None, 3)

Figure 5: Structure of the last layers in a Hierarchy model. We have cropped the diagram to fit into the document properly.

```
Evaluating the model over the testing set
376/376 [==============================] - 15s 37ms/step

Test loss: 0.06757183372974396
Test Class1_loss: 0.02565782144665718
Test Class2_loss: 0.023486744612455368
Test Class3_loss: 0.018427271395921707
Test Class1_mse: 0.03762137517333031
Test Class1_accuracy: 0.8711370229721069
Test Class2_mse: 0.0373929999768734
Test Class2_accuracy: 0.9260308146476746
Test Class3_mse: 0.030417753383517265
Test Class3_accuracy: 0.8349854350090027
```

This model generally learns to predict the classes with an accuracy above 80% and relatively small loss. No signs of overfitting have been detected (see Fig.6). However, it is not equally capable to distinguish all the classes. `Class1` and `Class7` suffer much more from the class imbalance issue, and this is clearly reflected in the capability of the model to classify the samples into their subclasses (see Fig.7).

Remarks & Observations

Let's summarize some other aspects we discover when experimenting with this type of models:

- The predictions should maintain the hierarchy imposed by the problem. Training the same model with mean squared error loss is generally better to do this. Numerically, the predictions are more similar to the test labels. Using the focal loss makes the sum of the probabilities for `Class1` to differ from 1.0 more (and more often). However, since we transform later the predicted probabilites to predicted labels to calculate the confusion matrices, what matters is that the maximum values are in the correct place. Focal loss then gives a model with slightly better perfomance metrics and generally sligthly less misclassified samples.
- The results analysis has to be done carefully. We have filtered the samples for each secondary class, creating masks based on the `Class1` predicitons. We used the same masks in both the

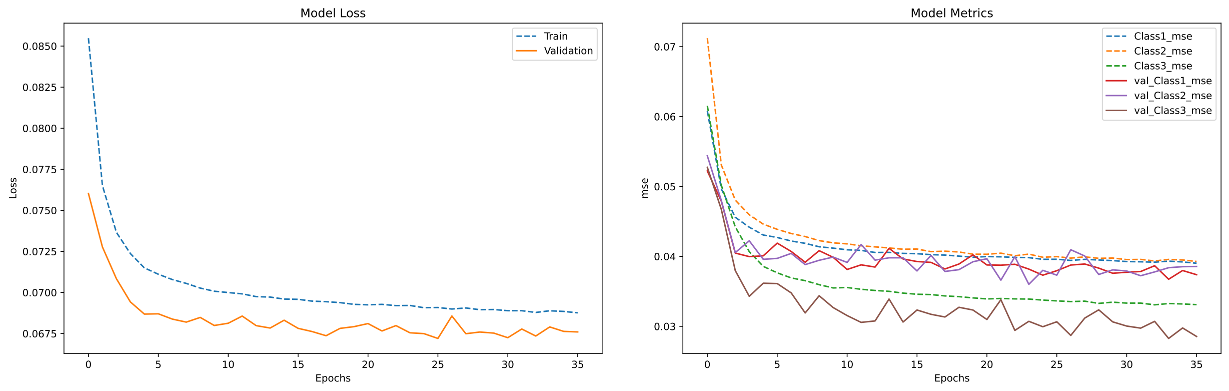

Figure 6: Hierarchy model history plots. On the left, the training and validation loss. On the right, the MSE metric for each of the classes.

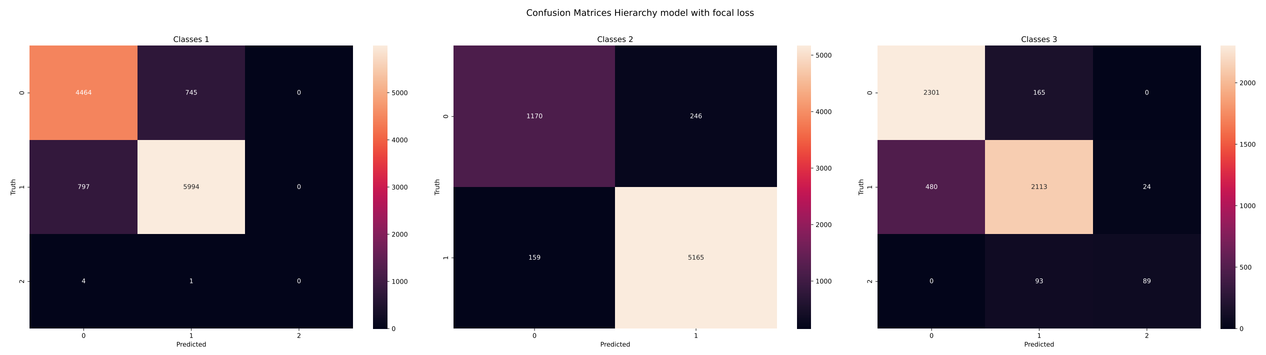

Figure 7: Confusion matrices of the hierarchy model for each of the classes.

test and predicted labels, which is not entirely accurate, but gives an approximation to the performance on each class.

3.3.2 Branch models

The second new family of models we have tried out receive the name of **branch** models. As described in Section 2.2, we take the *base* model architecture and modify it's structure by creating three branches with a single input and a multi-output type `Dense` layer at the end. Each branch has the same convolutional blocks and performs the correspondent convolutions separately. The `Class1` branch gets flattened and passed through the `Dense` blocks. Then, we create two masks with the relevant `Class1` outputs and concatenate them with the correspondent `Class2` and `Class7` branches after the flattening operation. `Lambda` and `Multiply` layers are introduced to enforce the hierarchy between the probabilities before building the final multi-output layer, which provides a list with an array for each of the classes (see Fig.8).

Similarly to the hierarchy models case, training of this model requires to use the custom loss functions and custom data augmentations generator we have designed. We have found that the best performing model is obtained using the focal loss, which helps a little bit with the class imbalance. We choose the focal loss parameters to be $\alpha = 0.2$ and $gamma = 1.5$ as in the hierarchy model case. Likewise, we do not use `Maxout` layers in any of the `Dense` blocks. Data augmentation parameters are the same as for the hierarchy model case (see Section 3.1.3.) We then trained the model implementing an EarlyStopping callback with patience = 5, which let's the model process the data for 29 epochs ($\sim$ 1h 44 min). The evaluation over the testing set yields:

```
  ----- EVALUATING MODEL -----
Evaluating the model over the testing set
376/376 [==============================] - 20s 52ms/step
```

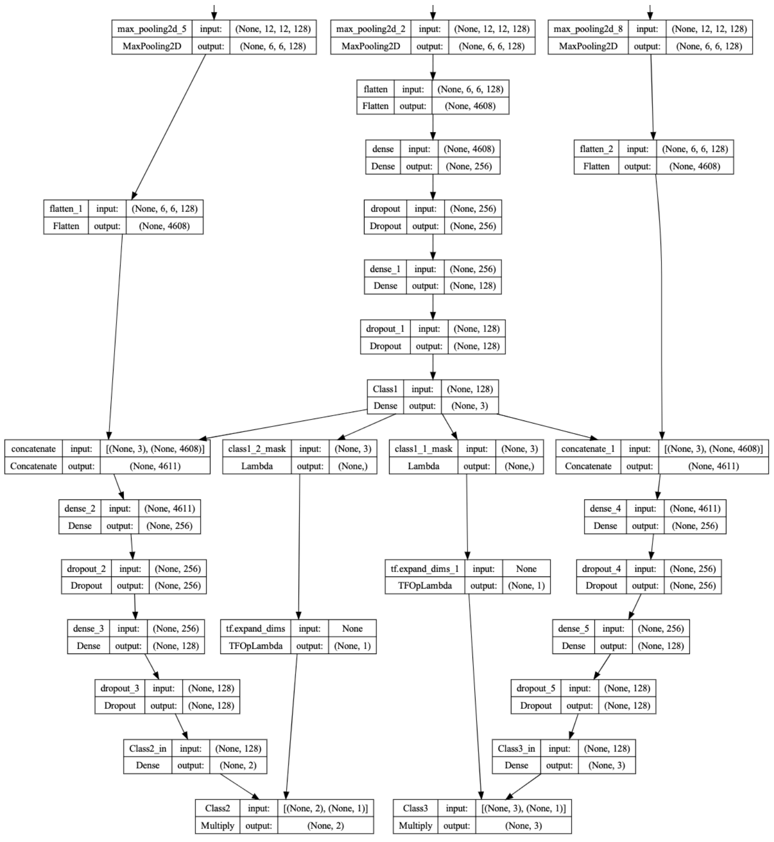

Figure 8: Branch model (partial) diagram. We have removed the rest of the convolutional blocks for simplicity and make it fit into the document

```
Test loss: 0.06945235282182693
Test Class1_loss: 0.027581214904785156
Test Class2_loss: 0.023605452850461006
Test Class3_loss: 0.018265655264258385
Test Class1_mse: 0.02979564294219017
Test Class1_accuracy: 0.8698042631149292
Test Class2_mse: 0.03684268146753311
Test Class2_accuracy: 0.9235318899154663
Test Class3_mse: 0.028385499492287636
Test Class3_accuracy: 0.856726348400116
```

This model generally learns to predict the classes with an accuracy above 85% and relatively small loss. No signs of overfitting have been detected (see Fig.9).

On the other hand, similarly to the hierarchy model, it is not equally capable to distinguish all the classes. `Class1` and `Class7` suffer much more from the class imbalance issue, and this is clearly reflected in the capability of the model to classify the samples into their subclasses. However, by looking at the confusion matrices, we conclude it performs better in this sense than the hierarchy model. This is clear by looking at the performance on `Class7` (see Figs.7 and 10).

Remarks & Observations:

Let's summarize some other aspects we discover when experimenting with this type of models:

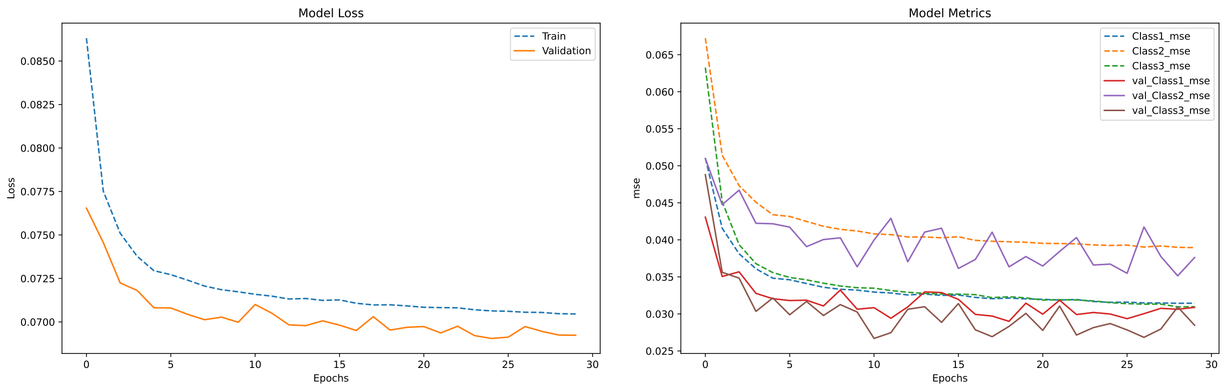

Figure 9: Branch model Losses and MSEs per epoch.

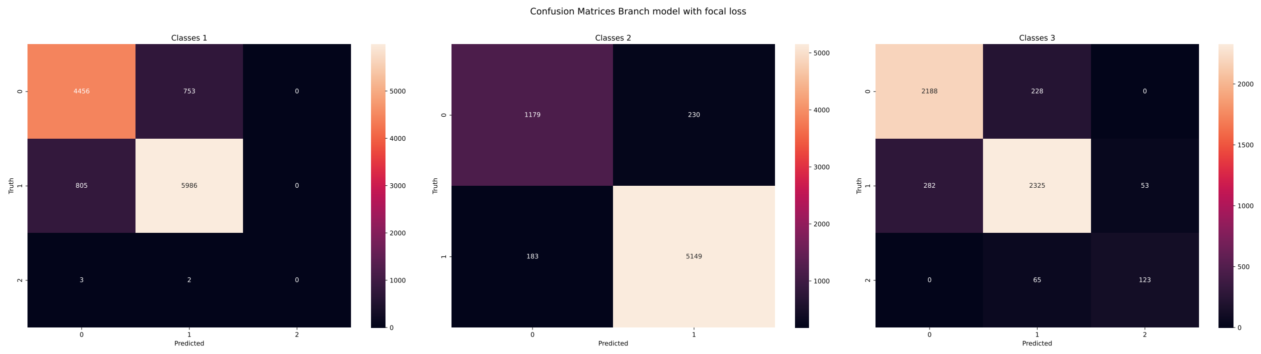

Figure 10: Confusion Matrices of the Branch model predictions for the different classes.

- The predictions should maintain the hierarchy imposed by the problem. In the Branch model cases, training them with mean squared error loss gives any advantage with respect to the focal loss. Numerically, the predictions match much better the conditions imposed by the problem. The sum of the predicted `Class1` probabilities is strictly 1.0 or very close to 1.0 in most of the cases. We observe that the condition on `Class2` and `Class7` is generally better preserved as well, but with less precision than for the `Class1` case.

- The results analysis has to be done carefully for the branch case as well. We predict the class probabilities, but we need to transform them to class labels first. We have filtered the samples for each secondary class, creating masks based on the `Class1` predicitons. We used the same masks in both the test and predicted labels, which is not entirely accurate, but gives an approximation to the performance on each class.

4 Conclusion

In this study, we addressed the challenging task of classifying galaxy morphology based on image data. Our work consisted of two main tasks: the classification of galaxies into primary categories and the hierarchical classification of subclasses within these categories. Through the utilization of convolutional neural network (CNN) models, we successfully achieved high accuracy in both tasks, providing valuable insights into the classification of galaxies.

For the first task, we focused on classifying galaxies into three primary categories: smooth, features or disk, and not a galaxy. Our CNN model, trained on a subset of samples with probabilities above 0.8, demonstrated exceptional performance with a global accuracy of 98%. The model's architecture consisted of convolutional blocks followed by dense blocks (`Maxout` + `Dropout` layers), culminating in an output layer with `softmax` activation. By employing regularization techniques such as dropout, data augmentation, and `EarlyStopping`, we effectively addressed the challenge of overfitting. Evaluation of the model on a testing set yielded a test accuracy of 98.4% and an F1-score of 98.4%, confirming the model's ability to accurately classify galaxies. The model's performance was further supported by the analysis of confusion matrices, which revealed minimal misclassifications among the primary categories. Overall, our findings highlight the efficacy of CNN models in accurately categorizing galaxy morphology.

The second task introduced an additional layer of complexity by incorporating hierarchical classification. Within the `Class1` subclasses, galaxies were further classified into `Class7` subclasses or `Class2` subclasses based on specific features. We explored two different model architectures to address this hierarchy: hierarchy models and branch models. The hierarchy models, modified versions of the base model, used `Lambda` and `Multiply` layers to enforce the numerical hierarchy between class probabilities. While these models achieved accuracy above 80% for each class, they encountered challenges related to class imbalance, particularly for `Class1` and `Class7`. On the other hand, the branch models divided the network into separate branches, enabling the independent classification of subclasses. By incorporating `Lambda` and `Multiply` layers to enforce hierarchy, the branch models exhibited improved performance compared to the hierarchy models. These models achieved accuracy above 85% for each class, demonstrating their ability to effectively classify subclasses within the hierarchical structure. An alternative would be to design custom loss functions that include this hierarchy directly, but we leave this experiment for future work.

Throughout our study, we leveraged various techniques to enhance model performance and mitigate challenges. The utilization of data augmentation, dropout regularization, and `EarlyStopping` proved vital in combating overfitting, leading to improved accuracy and generalization. Additionally, we explored custom loss functions, such as focal loss, to address class imbalance and encourage better performance on minority classes. We observed that class imbalance remained a persistent challenge, with certain subclasses experiencing lower accuracy. Future research could investigate advanced techniques, including ensemble models or transfer learning approaches, to further improve classification performance and overcome class imbalance issues.

Moreover, it is worth noting the role of ChatGPT, the language model utilized to assist in generating this study. ChatGPT, based on the GPT-4 architecture, provided valuable support in generating consistent code, content, assisting with the organization of the article, and providing additional insights into the tasks at hand. The model's ability to understand natural language and generate coherent responses facilitated the writing process and contributed to the overall structure and clarity of the article. However, it is important to acknowledge that the generated content should be carefully reviewed and validated by domain experts to ensure its accuracy and reliability.

In conclusion, our study demonstrates the effectiveness of CNN models in accurately classifying galaxy morphology. The successful classification of galaxies into primary categories and the hierarchical classification of subclasses provide valuable insights into the astrophysical processes at play. Despite the challenges posed by class imbalance, our models showcase remarkable performance and highlight the importance of architectural modifications, regularization techniques, and custom loss functions in improving classification accuracy. By further advancing the methodologies employed and exploring additional datasets, future research has the potential to enhance the generalization capabilities of these models and contribute to advancements in the field of astronomy.

Appendix

A. Focal Loss

Focal Loss is a loss function designed to address the problem of class imbalance in machine learning tasks, particularly in object detection and classification problems. It was introduced in [3, Lin et al.]

In many real-world scenarios, datasets often exhibit a significant class imbalance, where there are substantially more examples of one class than the others. This imbalance can lead to a learning bias towards the majority class, resulting in poor performance on minority classes. Focal Loss aims to mitigate this issue by assigning different weights to training examples based on their difficulty.

The focal loss function is defined as:

$$\mathrm{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

where p_t is the predicted probability of the true class for a given example, α_t is a balancing parameter that adjusts the weight assigned to the example, and γ is a focusing parameter that controls the rate at which the loss decreases as the predicted probability (p_t) increases.

The key insight behind focal loss is the introduction of the modulating factor $(1-p_t)^\gamma$, which down-weights the contribution of well-classified examples and focuses more on difficult-to-classify examples. By increasing γ, the loss function becomes more focused on hard examples, effectively addressing the class imbalance problem.

To further adjust the weight assigned to each example, the balancing parameter α_t is introduced:

$$\alpha_t = \begin{cases} \alpha & \text{if the true class is the minority class} \\ 1 - \alpha & \text{if the true class is the majority class} \end{cases}$$

Here, α is a hyperparameter that can be tuned to control the balance between the minority and majority classes. It allows the model to assign higher weights to the minority class, giving it more importance during training.

In a highly imbalanced dataset, the choice of α and γ depends on the specific characteristics of the dataset and the desired behavior of the model. Generally, it is recommended to set a higher value for α to upweight the minority class, ensuring that the model pays closer attention to correctly classifying examples from the minority class. For γ, a smaller value is often suggested (e.g., 1.0 or 1.5) to gradually decrease the loss for well-classified examples, focusing more on correcting misclassified instances. However, the optimal values should be determined through experimentation, considering the dataset and task requirements.

By using focal loss with appropriate values of α and γ, the class imbalance issue can be effectively addressed, and the model's ability to handle challenging examples can be enhanced.

References

[1] The Galaxy Zoo Project.
https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/about/research

[2] The Galaxy Challenge.
https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge

[3] Tsung-Yi Lin et a.l. *Focal Loss for Dense Object Detection.* https://arxiv.org/abs/1708.02002.