Università degli Studi di Milano

Data Science and Economics

Classifying galaxies with CNN

Student:
Michele Bartesaghi
Registration number: 17098A

ACADEMIC YEAR 2022-2023

Abstract

This project aims at creating a new categorical framework for the original regression task proposed by the Kaggle challenge GlaxyZoo. After comparing empirical results with a public submission, different possible classes were retrieved from the original regression data. After switching to categorical labels, multiple approaches were attempted in order to build an efficient classifier. This included reducing the number of classes initially designed (26), also to partially mitigate a severely problematic class imbalance. With 7 classes both a tiny *VGG-like* model and the architecture used for the regression task achieved approximately 70% of accuracy on the test data. Then, with model selection applied to tune the hyper-parameters of the *regression-like* architecture, an increase of approximately 2 percentage points was achieved. Finally, transfer learning with Xception model trained on ImageNet dataset was performed in a binary classification framework, thus achieving 79.1% of accuracy on the test set.

Contents

Chapter 1

Introduction

Figure 1.1: Hubble Ultra Deep Field

This picture celebrated its nineteenth anniversary a few days ago. The image is historical due to both its astonishing beauty and its scientific value. By looking at it, one can observe how galaxies appeared to be approximately one billion years after the Big Bang. Hence, tanks to this image one can look back in time, because observing the sky also means looking at the past. Apart from this fascinating side, such images have helped us in estimating the number of galaxies in our observable universe: two thousand billions.

The term "Deep Field" refers to pictures obtained with incredibly long exposure times in order for the weakest light sources to emerge.

This project aims at classifying galaxies on the basis of such images by deploying Convolutional Neural Networks (CNN).

The data used for this purpose come from the GalaxyZoo mission [**1**], which consists of giving common people the possibility to label galaxies, guided by a few questions about what they are observing in front of them. On the official website, one can read:

"To understand how galaxies formed we need your help to classify them according to their shapes. If you're quick, you may even be the first person to see the galaxies you're asked to classify.
Look at telescope images of distant galaxies.
Explore the sky. What will you find?"

After some research, I found out that both images and labels were collected in order to create a Kaggle challenge about this topic a few years ago.

The aim of the challenge was a regression task that will be better described in the following sections. As stated before, this project wants to come up with a classification framework instead of the original regression one, aiming at better interpretability.

Therefore, the ultimate goal of this project is not only to explore the possibility of substituting the "human in the loop", but also to study the topic in an innovative framework. Since this is a university project, priority is given to the deployment of proper techniques that lead to an efficiently trained model, with a good generalisation power.

Each one of the following sections corresponds to a notebook available on Google Colab, whose link will be written at the beginning of every chapter.

1.1 Decision tree

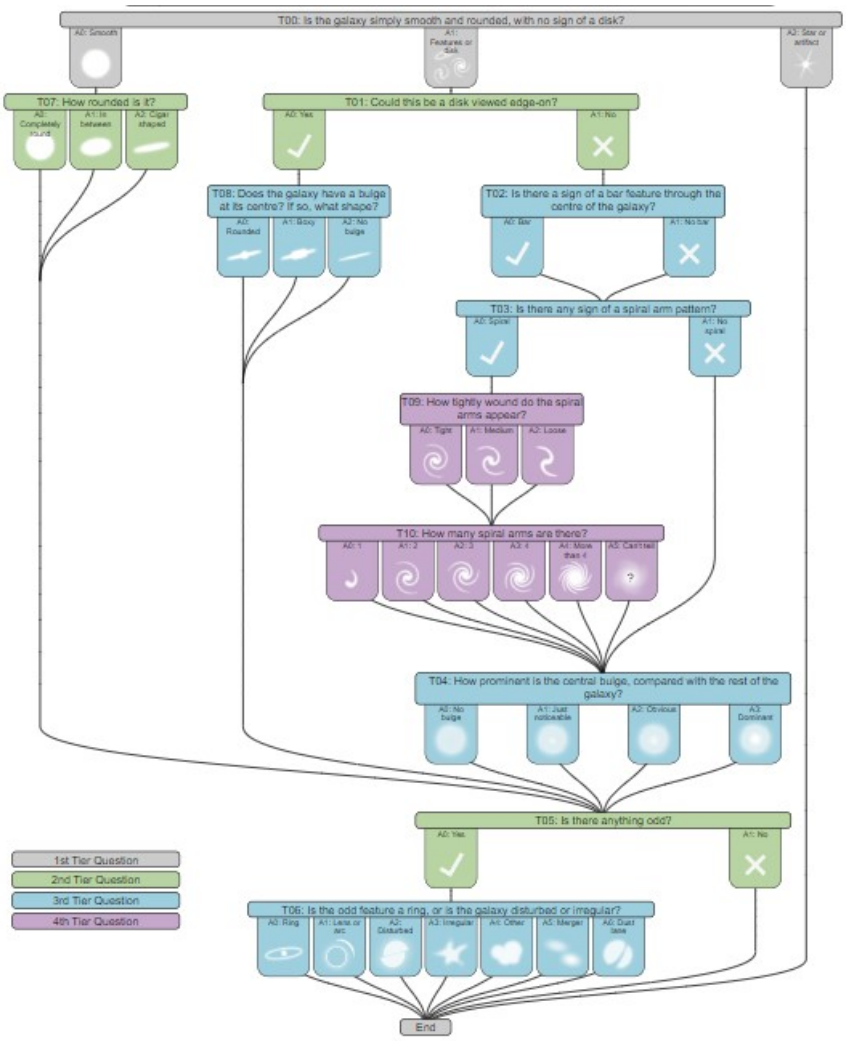

Figure 1.2: GalaxyZoo decision tree (morphology identification).

People who want to classify galaxies must answer 11 multiple choice questions created by researchers. Each possible response corresponds to a node in the decision tree 1.2. All the nodes have a probability distribution, in fact, for every question, the responses have a certain probability to be picked up. At the highest level (grey level) of the tree these probabilities are simply the likelihood of the galaxy falling in each category. Then, in order to emphasise that a good classification must get the "higher-level" morphology correct, probabilities are weighted after the 1st Tier Question as follows: for each subsequent question, the probabilities are first computed (these will sum to 1) and then multiplied by the value which led to that new set of responses (these will not sum to 1). One can think of this process as taking more into account the answer given at

the top, most important level: if that is right, details will be more relevant, otherwise even if details are correct, they will not be considered. For a better insight, consult the dedicated paper [**2**].

1.2 Data

As mentioned before, the data come from a Kaggle challenge posted in 2013, named Galaxy Zoo - The Galaxy Challenge. The data provided by this challenge mainly consist of 3 different folders:

- images_training: JPG images of 61578 galaxies, named according to their *GalaxyID*.
- solutions_training: Probability distributions for the classifications for each of the training images.
- images_test: JPG images of 79975 galaxies, named according to their *GalaxyID* (This folder was intended for the submission of the results and was not used for this project).

The images size is 424x424, which is considerably high and would require much more computational power than the one at disposal (see figure 1.4). Moreover, each picture contains multiple galaxies, whereas the focus must be on the central one. As for the solutions_training file, it is a csv file where the first column is labelled *GalaxyID*, which allows to match each row with the corresponding image. Then there are 37 more columns containing floats in $[0, 1]$, which represent 37 different *morphological* classes a galaxy can belong to. Hence, as mentioned before, the morphologies are represented by probabilities of belonging to that category: a float close to 1 stands for many users identifying this feature for the considered galaxy with a high level of confidence.

The shape of the dataframe is: (61578, 38)

	GalaxyID	Class1.1	Class1.2	Class1.3	Class2.1	Class2.2	Class3.1	Class3.2	Class4.1	Class4.2	...	Class9.3	Class10.1	Class10.2	Class10.3	Class11.1	Class11.2	Class11.3	Class11.4	Class11.5	Class11.6
0	100008	0.383147	0.616853	0.000000	0.000000	0.616853	0.038452	0.578401	0.418398	0.198455	...	0.000000	0.279952	0.138445	0.000000	0.000000	0.092886	0.0	0.0	0.0	0.325512
1	100023	0.327001	0.663777	0.009222	0.031178	0.632599	0.467370	0.165229	0.591328	0.041271	...	0.018764	0.000000	0.131378	0.459950	0.000000	0.591328	0.0	0.0	0.0	0.000000
2	100053	0.765717	0.177352	0.056931	0.000000	0.177352	0.000000	0.177352	0.000000	0.177352	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000
3	100078	0.693377	0.238564	0.068059	0.000000	0.238564	0.109493	0.129071	0.189098	0.049466	...	0.000000	0.094549	0.000000	0.094549	0.189098	0.000000	0.0	0.0	0.0	0.000000
4	100090	0.933839	0.000000	0.066161	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000

Figure 1.3: Labels from solution_training file.

To give an insight into the classes, what follows is a concise textual explanation :

- Class 1 (**Shape**):
  - .1: Smooth, .2: Features or disk, .3: Star or artefact
- Class 2 (**Disk-like**):
  - .1: Disk viewed edge-on, .2: No
- Class 3 (**Bar through the centre**):

  - .1: Bar, .2: No bar
- Class 4 (**Spiral**):
  - .1: Spiral, .2: No spiral
- Class 5 (**Bulge**):
  - .1: No bulge, .2: Just noticeable, .3: Obvious, .4: Dominant
- Class 6 (**Anything odd**):
  - .1: Yes, .2: No
- Class 8 (**Odd feature**):
  - .1: Ring, .2: Lens or arc, .3: Disturbed, .4: Irregular, .5: Other, .6: Merger, .7: Dust lane
- Class 7 (**Roundness**):
  - .1: Completely round, .2: In between, .3: Cigar shaped
- Class 9 (**Bulge shape**):
  - .1: Rounded, .2: Boxy - .3: No Bulge
- Class 10 (**Spiral arms**)
  - .1: Tight, .2: Medium, .3: Loose
- Class 11 (**Spiral arms number**)
  - .1: 1, .2: 2, .3: 3, .4: 4, .5: More than 4, .6: Can't tell

With all this being said, one can clearly see that the original problem was a regression task. Before developing the classification task, I emulated one of the results obtained by a public submission (with subtle differences) in order to get acquainted with input pipelines and with the fashion things work in case of a regression with CNN. For more details see the appendix A.

Figure 1.4: Original images.

Chapter 2

Categorical Framework

The corresponding notebook is available at this link.

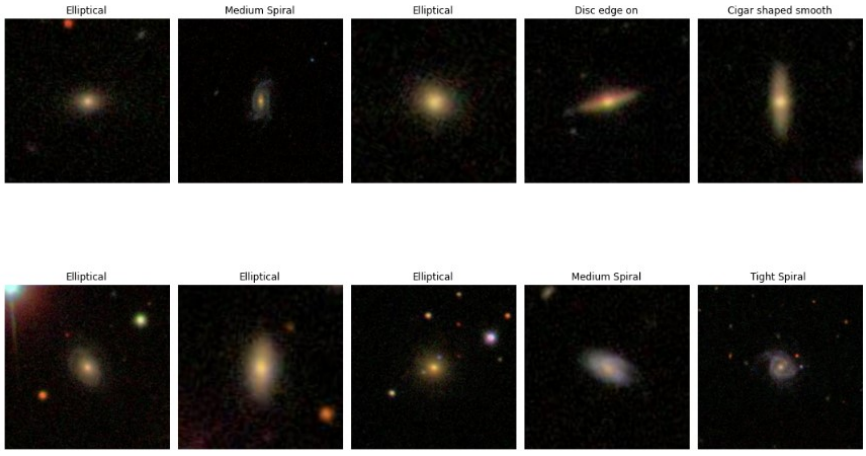

Figure 2.1: Original images with categorical labels.

Starting from the textual description of the labels provided above, I decided to create a *coarse-grained* custom function to transform the problem into a classification task with categorical labels, reducing the number of classes from 37 to 26, also for interpretability purposes. The idea behind this function was just to follow the various possible paths of the decision tree and assign a label on the basis of the highest morphologies scores. Later on, this approach revealed to be ineffective, mainly because of a tremendous class imbalance issue, which can be observed in figure 2.4. Notice also that with an elevate number of highly imbalanced classes the probability of having all the distinct labels in one batch is low and training gets increasingly difficult. Attempts to use a custom weighted categorical crossentropy loss were made, but the results were shockingly unacceptable. Then I reduced the number of classes, inspired by the Hubble classification fork, and attempted to use SMOTE both with *strategy minority* and *not majority* to oversample the less numerous class and all the minor classes respectively. Regardless of how promising the setting was, results were totally unsatisfactory.

A possible solution to the unbalanced dataset problem, requiring a low number of classes, will be discussed in the Model Selection section.

Finally, there is a relevant issue that could be tackled in future works: the boundary between, for example, an In Between Round Smooth galaxy and a Disc galaxy is really subtle. Even the volunteers participating into the GalaxyZoo mission have a hard time trying to correctly label them, causing labels to be really noisy. Therefore the question of how to underline the differences between them, maybe with a more accurate preprocessing, is definitely worth further studying.

Figure 2.2: Galaxy classified as (Smooth) Cigar Shaped with high confidence.

Figure 2.3: Galaxy classified as Disc with high confidence.

With that being said, these are the final classes used for this project:

1 - Elliptical

2 - Cigar shaped smooth

3 - Disc edge on

4 - Tight spiral

5 - Medium spiral

6 - Loose spiral

7 - Other

By removing the majority of classes, one could expect to leave a considerable number of galaxies unclassified. Surprisingly, in this case only 64 images were left unlabelled, which means approximately 99.999% of the original data is still available. Then, all the unclassified galaxies were removed from the folder in order not to encounter issues later on. In figure 2.1 one can see a few examples of images with the corresponding categorical labels.

Figuring out how to deal with this unbalanced dataset and setting a proper framework was the most challenging part, because it basically included a lot of trial and error.

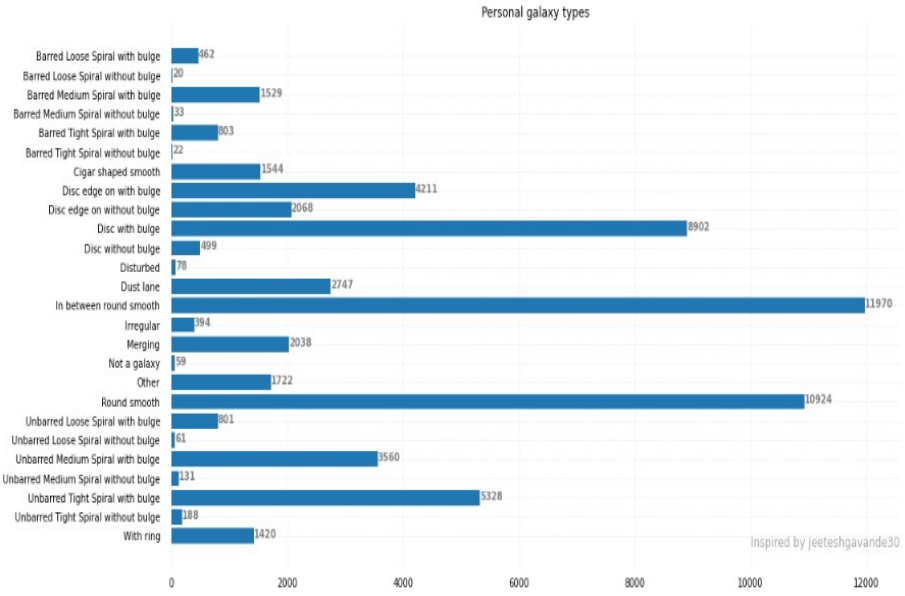

Figure 2.4: 26 class frequencies

2.1 Preprocessing

After creating the categorical labels dataset, I opted for OneHot encoding the labels in order to use the categorical crossentropy as a loss function and categorical accuracy as a metric, sticking to what appear to be the state-of-the-art of classification and most common used tools. However, there are multiple other options. For example, one might decide to encode the labels as integers and make the CNN output a single float that, hopefully, is closer to the actual integer label of the galaxy being classified[1].

In the different notebooks I adopted different techniques to prepare both the datasets and the images.

In both the basic and the Transfer Learning framework efficient input pipelines were necessary to make the training faster and more effective when dealing with innumerable, relatively high-dimensional examples. Therefore, during the preprocessing part I adopted TensorFlow pipelines to make it easier to handle such amount of data, namely performing transformations on images, combining them with labels in a single dataset, batching, shuffling and prefetching the latter. By reading the notebook one can see these concept in details, explained by a quite detailed commentary.

When performing model selection I opted for ScikitLearn module because I was using less data and it is more straightforward when it comes to coding and accessing elements. On top of that, I used GridSearchCV, which comes from the same module.

In the basic framework I retained the central portion of each image (150x150) in order to exclude additional irrelevant information. Then I normalised the

[1] This can be achieved through the use of Keras loss function and metric SparseCategoricalEntropy() and SparseCategoricalAccuracy() respectively.

images and rescaled them to (50,50,3), where 3 is the number of color channels (RGB), to save computational time.

At the end of the preprocessing I had my data split in training set (80% since in DeepLearning a lot of data is required to train the models), validation set (17%) and test set (3%, containing about 2 thousand images, that are enough to assess the power of generalisation of the model), with stratified labels, namely the relative frequency of each class in any of the three splits was identical to the relative frequency in the whole dataset.

2.2 Callbacks

In order to make the training more efficient the following callbacks were mainly used:

- ReduceLROnPlateau: it monitors the validation loss and if it is not decreasing the learning rate is reduced by a $factor = 0.2$ after $patience = 3$ epochs.
- LearningRateScheduler: even though there is not a plateau, the learning rate is reduced every 10 epochs by the 20%
- EarlyStopping: it monitors the validation loss and stops the training if there is no improvement with respect to the achieved minimum after $patience = 7$ epochs

Observe that it is crucial to set the learning rate just right, in order neither to miss the optimal point nor to be stuck around a particular one (maybe a local minimum). All the models were trained for 25 epochs with Stochastic Gradient Descent (SGD), because it appeared to perform better than Adam optimiser on this task.

2.3 VGG-like model

```
Model: "TinyVGG"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_142 (Conv2D)         (None, 48, 48, 64)        1792

 conv2d_143 (Conv2D)         (None, 46, 46, 64)        36928

 max_pooling2d_74 (MaxPoolin  (None, 23, 23, 64)       0
 g2D)

 conv2d_144 (Conv2D)         (None, 21, 21, 32)        18464

 conv2d_145 (Conv2D)         (None, 19, 19, 32)        9248

 max_pooling2d_75 (MaxPoolin  (None, 9, 9, 32)         0
 g2D)

 conv2d_146 (Conv2D)         (None, 7, 7, 16)          4624

 conv2d_147 (Conv2D)         (None, 5, 5, 16)          2320

 dropout_42 (Dropout)        (None, 5, 5, 16)          0

 flatten_36 (Flatten)        (None, 400)               0

 dense_42 (Dense)            (None, 7)                 2807

=================================================================
Total params: 76,183
Trainable params: 76,183
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.5: VGG-like model

After a lot of trial and error, including large and complex architectures performing poorly, I opted for a *VGG-like* architecture (2.5). I trained it with the aforementioned loss, metric, callbacks and optimizer. Observe also that *valid padding* was used because after cropping the images, there is no clear prevalence of black pixels around the galaxy. This parameter ensures that not excessive spatial information is lost. The training took 2.5 hours and the model achieved an accuracy of 69.8% on the test set, which is bad, but not totally unsatisfactory, taking into account that classes are incredibly skewed.

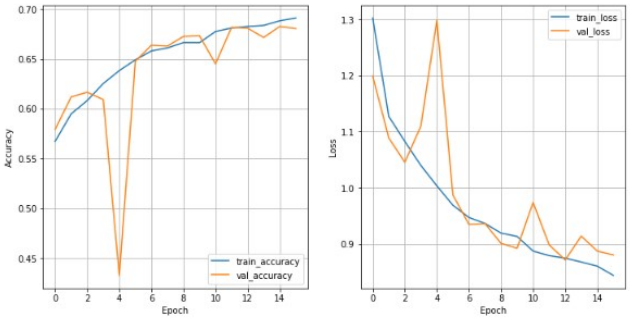

Figure 2.6: Learning curves. There seems to be a decreasing trend, but just in the training accuracy. With more epochs the model is just likely to badly overfit

2.4 Regression-like model

Since the regression task was carried out quite successfully by the CNN architecture described in the Appendix A, I used the same training setting to train that model and assess whether it performs well on the classification task or not. Note that also in this case I adopted the *valid padding* approach. The training took approximately 1 hour and the result was a 70.5% of accuracy on the test set. By looking at the learning curves in the picture below, one can see that the training loss is almost monotonically decreasing. If the model had been trained with more epochs, it might have increased its performance, but it might have started badly overfitting as well. Despite the class imbalance, this is a great improvement with respect to the innumerable attempts preceding this one. Finally, not only balancing classes, but also using higher resolution images might have led to a better result, because the amount of details required for this kind of classification is intuitively really high. With these ideas in mind, I moved onto the next experimental result, explained in the following section.

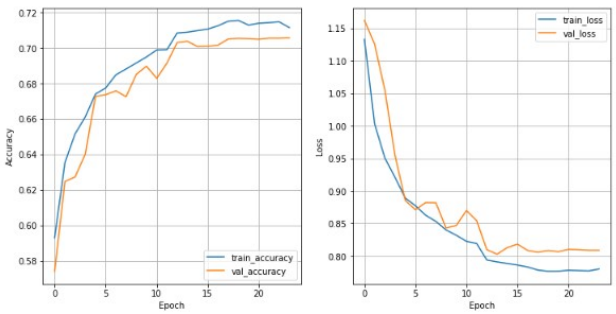

Figure 2.7: Plotting curves of the model used for the regression task.

2.5 Model selection

The corresponding notebook is available at this link.

In this notebook I wanted to deal with two main aspects:

- Class imbalance
- Model selection for hyper-parameters tuning

First and foremost, I only worked with a random subsample of 20% of the available images for time and memory consumption reasons. I retained the central portion of each image and resized it to (100,100,3) after normalising it. I adapted the function to set categorical labels in order to retrieve 3 classes only: I wanted to understand the possibility of distinguishing at least between elliptical, spiral and other kind of galaxies (irregular e.g.). This solution led to the loss of 1% of the original images. Almost 65% of the remaining galaxies falls in the elliptical class. However, having only three classes allowed me to use the *class_weight* parameter during the model training. This solution tells the model to penalise more mistakes on the less numerous classes, mitigating the effect of the labels skewness on the final result. The weights were computed as follows

$$\text{weight}_i = \frac{t}{cf_i}$$

where t is the total number of examples, c is the number of classes and f_i is the frequency of the i-th class. Hence, the weight for the Elliptical class was 0.513, the weight for the Spiral class was 1.321 and the Other class had a weight of 3.426. I split the data into training set (70%), validation set (21%, in proportion bigger than before to have a slightly more accurate estimation of the generalisation power) and test set (9%), stratifying labels. Since the architecture of the so-called *Regression-like* model was outperforming the *VGG-like* CNN, I decided to stick to it. However, this model makes use of dropout regularisation layers in order to prevent it from overfitting. To tune the dropout rate among the fully connected layers, as well as the batch size[2], I run a GridSearch on the training set: the parameters of the model used to apply the fit are optimized by cross-validated grid-search over a parameter grid. I performed the search over the following restricted search space: dropout rate= $[0.1, 0.3, 0.4]$ and batch size = $[32, 64, 128]$, using 3 epochs, because GridSearchCV() is a tremendously demanding algorithm both in terms of computational time and memory usage. The idea was to spot the combination of parameters leading to the best learning trend over the first epochs. Of course the search can be performed on many more hyper-parameters, including for example the optimizer and the loss function. I run the algorithm multiple times (also with different grids) and the best score over three epochs was on average 60% achieved with batch size of 32 and dropout rate 0.3. Therefore I build the model with the selected hyper-parameters and trained it in a second moment, using 50% of the original images[3]. The training took approximately 1.5 hour with an early stop at epoch 21. Subsequently, the

[2] The learning rate was already controlled by callbacks.

[3] Not shockingly, the model achieved a worse result when trained with less data.

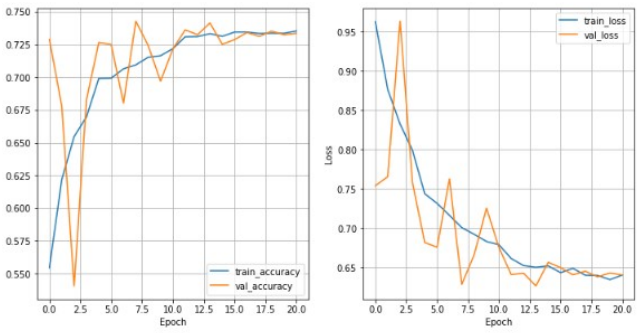

Figure 2.8: Learning curves of the selected model. Thanks to the dropout there is no overfitting, but the accuracy in general is still low.

model achieved an accuracy of 72.8% on the test set[4]. The improvement is worse than expected, but by taking a look at the misclassified images one can get an idea of which features the model is struggling to extract, and develop further on this point, as suggested in the next section.

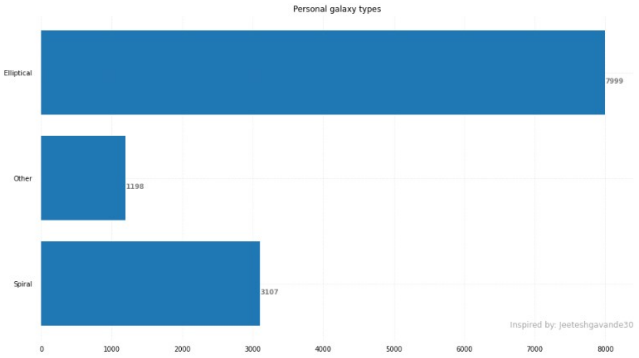

Figure 2.9: Distribution over 3 classes

2.6 Transfer Learning

The corresponding notebook and the trained architecture file are available at this link and this link respectively.

The idea was to deploy notoriously efficient CNN architectures, trained on extremely large datasets (ImageNet e.g.) to increase the accuracy that could be achieved in this categorical framework. The use of already trained architectures, adequately adapted to a specific situation, is known as Transfer Learning ([**4**], *"deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning".*). I tried with different architectures, namely VGG16, DenseNet121, InceptionV3, trained on ImageNet dataset, initially with the 26-classes design and after with

[4]On average when trained on less data it managed to achieve around 69 per cent only, or 70 per cent at most.

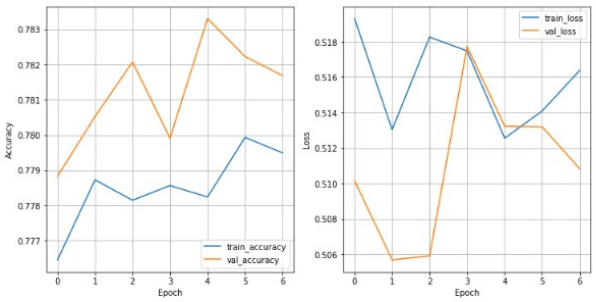

Figure 2.10: Learning curves of the 11th round of training.

the 7-classes setting; either settings provided bad results. Then I thought of starting to work on binary classification, in order to teach the model how to extract peculiar features that allow to discriminate between elliptical and *non-elliptical* ones (cutting at the first level of the tree) [5].

Moved by this idea, I thought of testing the effectiveness of transfer learning in this binary classification framework. Hence the two categorical classes were encoded as Smooth, that stands for the leftmost node in the first level of the decision tree, and With features/Disk, essentially representing anything else (starting from the second and third node in the 1st Tier Question). During the preprocessing the central portion of the images was retained they were resized to work with (100,100,3) pictures, both because this might contributed to the improvement during model selection and due to the fact that most of the pre-trained models require higher-dimensional input. I opted for the Xception model trained on the ImageNet dataset. For details about its architecture see [**4**]. After setting the *base* model as non trainable, I built the customised model upon that one. The latter includes the input layer, a scale layer to normalise images after a layer devoted to data augmentation, which randomly flips pictures to increase the power of generalisation of the model. Then, after the Xception network, there is a flatten layer, followed by a dropout layer, with 20% of probability for a neuron to be excluded during training, and the output layer. This last layer consists of two neurons, because we are now working with two classes only, with softmax activation function. The trainable parameters are just approximately 30 thousands instead of almost 21 millions, having the base model "frozen". I trained the model for 12 rounds of 10 epochs each with Adam optimiser with a really low learning rate (10^-5). In general, the model seemed to underfit during every round, but this might be due also to the dropout and data augmentation layers, which are not used during validation. However, the model managed to achieve the highest test accuracy, which was 79.1%, after the 11th round. In the last round it showed no improvement and ended with an early stop during the 6-th epoch. The total training took about 24 hours.

[5] I had started something in this sense in an additional notebook which has not been included in the project, intended for future developments of this work. An ad-hoc weighted categorical crossentropy, combined with a better understanding on how to extract the right features, could be the key to solve the original multi-class categorical problem that inspired this project

2.7 Future works

In future works it could be interesting to apply transfer learning to the multi-class problem stated at the beginning of the project, with a different learning rate, better controlled by callbacks, and including rounds of fine tuning of the model, which could lead to significant improvements. Is it possible to reach 80% of accuracy on the test set?

Figure 2.11: Images misclassified with transfer learning.

Chapter 3

Considerations

This project presented a few challenges that are worth to be underlined: dealing with a large amount of data (2 GB compressed), working with images that are all rather similar to each other, facing a staggeringly imbalanced dataset and setting up a framework which does not seem to have been exhaustively studied previously. The results of the different models are very distant from an acceptable performance, but they are a starting point for a more elaborated and challenging work. The best strategy seems to be the architecture proposed also in the regression framework trained with SGD, which achieved around 72% of accuracy on three classes, with a weighted loss to mitigate the class imbalance, and 70% on seven classes. Finally, thanks to transfer learning, it was possible to achieve almost 80% accuracy on the test set, in a problem of binary classification.

Figure 3.1: Milky way over ALMA telescope.

Appendix A

Regression model

The corresponding notebook is available at this link.
For the regression model reproduction I give credits to the Kaggle user Ushmi.

A.1 Preprocessing

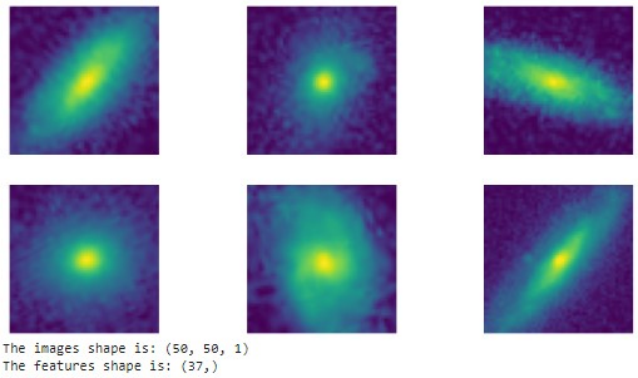

Figure A.1: Processed images contained in the training set.

A few modifications were made during the preprocessing. The reason for such changes is to assess whether some aspects appears to be relevant or not. In particular, I was interested in the impact of image resize and normalisation, as well as greyscale conversion. Neither of these techniques were presented in the aforementioned submission, therefore this approach could lead me into the image processing part of my project. Notice that I also opted for a more "aggressive" resizing ((50,50,3) instead of (100,100,3)) with the only purpose of cutting down computational time. Finally the same TensorFlow pipelines described in previous sections were used to make the training even more efficient[1]. Notice that during this step I did not pay attention to labels frequency, because that would be quite complicate.

[1] In order to speed up the process, I only used 20% of the images, drawn at random

A.2 Model

After creating the training (80% of the data), validation (70% of the remaining) and test (30% of the remaining) set with TensorFlow, I reproduced the model and trained it, using the same callbacks described before to make the training process more efficient. The *Regression_model* has about 300 thousands trainable parameters. We could call a *pack* a set of layers consisting of a convolutional layer, a BatchNormalisation layer, a MaxPooling layer and a Dropout layer in this order. In the model 4 packs with increasing number of filters and decreasing window sizes can be found before flattening and adding 3 fully connected layers with dropouts (figure A.3). The i-th output node produces a float between 0 and 1, which is the probability for the galaxy of having the morphological structure inquired by the i-th node. Furthermore, observe that since it is a regression task the loss function is the extensively used MSE, whereas the deployed metric is the RMSE, which is preferred because it has the same units as the response variable. The model was fitted using an Adam optimiser with a decreasing learning rate for 25 epochs. In figure A.2 one can spot a decreasing trend, but also a tendency to overfit. Since the main purpose of the project is not a regression task, I did not elaborate on this point.

Results are neither contemptible nor exceptional, keeping in mind that the model achieved a RMSE of 0.112 on the test set, which would result in a leaderboard position within the top 100 best scores (the lowest RMSE achieved was 0.074).

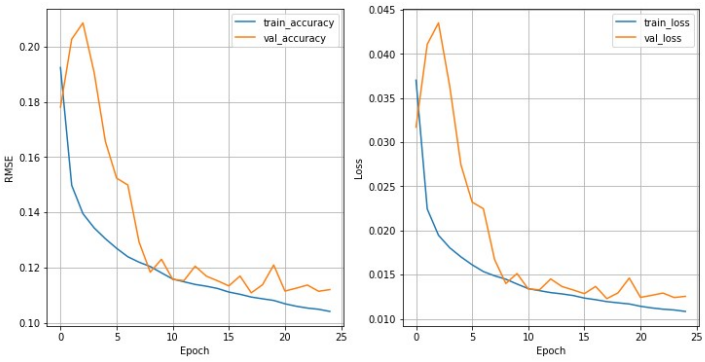

Figure A.2: Learning curves.

```
Model: "Regression_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_16 (Conv2D)          (None, 48, 48, 16)        160

 batch_normalization_16 (Bat  (None, 48, 48, 16)       64
 chNormalization)

 max_pooling2d_16 (MaxPoolin  (None, 24, 24, 16)       0
 g2D)

 dropout_24 (Dropout)        (None, 24, 24, 16)        0

 conv2d_17 (Conv2D)          (None, 20, 20, 32)        12832

 batch_normalization_17 (Bat  (None, 20, 20, 32)       128
 chNormalization)

 max_pooling2d_17 (MaxPoolin  (None, 10, 10, 32)       0
 g2D)

 dropout_25 (Dropout)        (None, 10, 10, 32)        0

 conv2d_18 (Conv2D)          (None, 8, 8, 64)          18496

 batch_normalization_18 (Bat  (None, 8, 8, 64)         256
 chNormalization)

 max_pooling2d_18 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)

 dropout_26 (Dropout)        (None, 4, 4, 64)          0

 conv2d_19 (Conv2D)          (None, 2, 2, 128)         73856

 batch_normalization_19 (Bat  (None, 2, 2, 128)        512
 chNormalization)

 max_pooling2d_19 (MaxPoolin  (None, 1, 1, 128)        0
 g2D)

 dropout_27 (Dropout)        (None, 1, 1, 128)         0

 flatten_4 (Flatten)         (None, 128)               0

 dense_12 (Dense)            (None, 512)               66048

 dropout_28 (Dropout)        (None, 512)               0

 dense_13 (Dense)            (None, 256)               131328

 dropout_29 (Dropout)        (None, 256)               0

 dense_14 (Dense)            (None, 37)                9509

=================================================================
Total params: 313,189
Trainable params: 312,709
Non-trainable params: 480
_________________________________________________________________
```

Figure A.3: CNN model for regression task.

Bibliography

[1] GalaxyZoo website:
https://www.zooniverse.org/projects/zookeeper/galaxy-zoo/

[2] Kyle W. Willett, Chris J. Lintott, Steven P. Bamford, Karen L. Masters, Brooke D. Simmons, Kevin R. V. Casteels, Edward M. Edmondson, Lucy F. Fortson, Sugata Kaviraj, William C. Keel, Thomas Melvin, Robert C. Nichol, M. Jordan Raddick, Kevin Schawinski, Robert J. Simpson, Ramin A. Skibba, Arfon M. Smith, Daniel Thomas, Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey, *Monthly Notices of the Royal Astronomical Society*, Volume 435, Issue 4, 11 November 2013, Pages 2835–2860, https://doi.org/10.1093/mnras/stt1458

[3] F. Sultana, A. Sufian and P. Dutta, "Advancements in Image Classification using Convolutional Neural Network," *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, Kolkata, India, 2018, pp. 122-129, doi: 10.1109/ICRCICN.2018.8718718.

[4] Keras Applications