Galaxies Recognition with ML

Artificial Intelligence and Machine Learning Course Project - Politecnico di Torino

Alberto Butera
s265362

Riccardo Gabellone
s256793

Giuseppe Como
s267608

Domenico Flavio Amato
s265560

Abstract

This document describes our solution for the Galaxy Challenge on Kaggle relating the classification of galaxies. Taking a dataset of images about galaxies and a decision tree made by the feedback of hundreds of thousands of volunteers, we built a system based on Convolutional Neural Network which not only performs image classification but also image prediction. Our proposal focuses on finding the best Network's set of layers by evaluating the predictive performance by Root Mean Squared Error (RMSE) metric and aims to predict features' probabilities for new galaxy images. Prediction's results have been rendered more comprehensible by adding descriptive labels next to each image. The final result was achieved using a blending of several variations of Neural Network.

1. Introduction

From the dawn of times, thirst of knowledge moved humans to ask themselves existential questions. Several of them were directed to the sky: humans have been using stars to navigate, predict rain, get time and seasons. One of the crucial points in this field is to understand how and why we are here and part of the answer to this question lies in the origins of galaxies, such as the Milky Way.
There are questions which still exists today towards Milky Way and other billions of galaxies present in our Universe such as how they have formed or how they have evolved. Having a look at the photos, we can see as each galaxy appears to us in all its splendor showing off different shapes like beautiful spirals or huge ellipticals, various sizes and gaudy colors.
To solve this puzzle we have to find critical pieces by understanding distribution, location and type of galaxies and studying their main features like shape, size and color.
As more powerful astronomical instruments were brought to us, we are becoming capable to push our sight beyond human limits, moreover telescopes are collecting every passing day more and more images of distant galaxies making datasets very huge. In order to better understand how the morphology of galaxies relate to the physics that create them, such images need to be sorted and classified.
Our project's purpose is to help in providing a solution which takes as input an image-based dataset with a target containing weights (it is more a regression than a classification). Weights come from answers of a number of users to several questions about galaxies and their morphology, indeed each query is associated to a class in the target and each weight corresponds to the percentage of users who answered with a positive result to the respective question. Furthermore, our project is able to make predictions by means of a decision tree (fig.1), classifying records by assigning them to the most likely class.

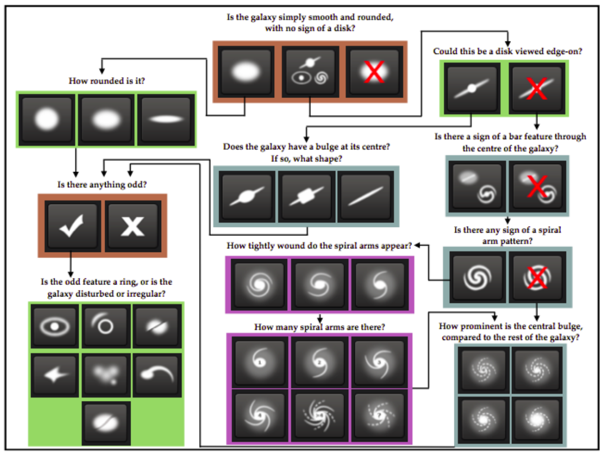

Is the galaxy simply smooth and rounded, with no sign of a disk?
Could this be a disk viewed edge-on?
How rounded is it?
Does the galaxy have a bulge at its centre? If so, what shape?
Is there a sign of a bar feature through the centre of the galaxy?
Is there anything odd?
Is there any sign of a spiral arm pattern?
How tightly wound do the spiral arms appear?
Is the odd feature a ring, or is the galaxy disturbed or irregular?
How many spiral arms are there?
How prominent is the central bulge, compared to the rest of the galaxy?

Figure 1. Flow chart of the decision tree, beginning at the top center

The most popular architecture used for image classification is Convolutional Neural Network where the image is passed through a series of convolutional, nonlinear, pooling layers and fully connected layers, and then generates the output. That is the reason why our choice fell on this type of architectures, changing layers' set of the network for every attempt we performed. Due to the size of the dataset and the amount of layers introduced, execution times became quite high, but Google Colab environment, which offers GPU acceleration, has allowed us to speed up experiments.

2. Data preparation

Initially, we created a repository on Github where uploading all the material needed and saving all the different attempts tried with respective results. So we downloaded the dataset provided from Kaggle about the Galaxy Zoo challenge and uploaded it to the repository created before [1].
The initial dataset was made up of images 424*424 with 3 color channel (RGB) divided into two folders: *images_training_rev1* with 61578 images and *images_test_rev1* with 79975 images.
For data and network preparation we used Keras, that is a high-level neural networks API capable of running on top of TensorFlow [2].

2.1. Preprocessing and Data Augmentation

During the preprocessing phase, we generated batches of tensor image data with real-time data augmentation, one for train dataset and one for test dataset, using *ImageDataGenerator* class from Keras.
In this case, "real-time" data augmentation is done during the training phase, in fact this reduces overfitting because the network would never see the same image twice since the training data is randomly augmented at every epoch.
Galaxy images are rotation, scale and translation invariant, so the convolutional neural network can classify images even when they are placed in different ways. This is useful to have "more data" to train the model and to prevent overfitting, so we decided to exploit them to do data augmentation.
We set the following parameters values:

- **rotation**: a random rotation of an angle of 90 degrees;
- **translation**: a random shift between -0.1 and 0.1 pixels for both width and height;
- **flip**: horizontal and vertical reflection;
- **rescale**: multiply the values of data by 1/255 scaling them between 0 and 1, because they are too high for our model to process;

We imported *training_solutions_rev1.csv* from our Github repository and using *flow_from_dataframe* we created train generator and valid generator that are respectively 90% and 10% of the entire training dataset.
For both, we defined dataframes that contain the provided weights, source folder *images_training_rev1*, a random seed for shuffling and transformations, the subset respectively "training" and "validation", the *batch_size* and the *target_size* that resize the images to 224 x 224 pixels in order to fit the input size of our network architecture.

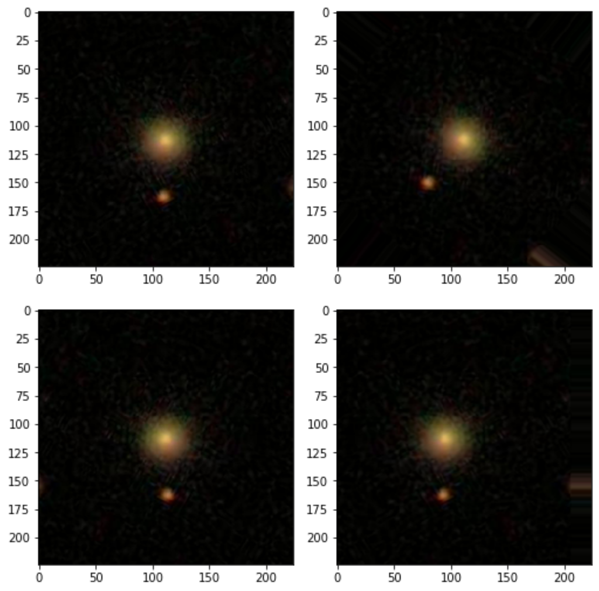


Figure 2. Transformations examples (left to right, top to bottom): original, rotation, flip, shift

3. Network

As regarding the network, we found our best solution after many attempts because we wanted to face as good as possible the challenge proposed in the now-concluded competition.
Our solution was about CNN (Convolutional Neural Network) that let us to better differentiate and calculate the multiple weighted features from the source images. We used the one with the best trade-off between loss and computational time according to our hardware possibilities.

3.1. Architecture

We chose a convolutional neural network architecture based almost entirely on Xception, which stands for "Extreme Inception", a combination of depthwise separable convolution layers. Its architecture has 36 convolutional layers forming the feature extraction base of the network. These layers are structured into 14 modules, all of which have linear residual connections around them, except for the first and last modules [3].
Nextly we added the dense part that is based firstly on a flatten layer that does not affect the batch size but provided us a tensor to reduce all the dimensions to one (2048x7x7 = 100352). Then we used a dropout which helps prevent overfitting by randomly setting a 0.25 fraction rate of input units to 0 at each update during training time. At last, we put the dense layer with the outputs of the final 37 weights. All of this made our architecture very easy to define and

modify: it takes only few lines of code using a high-level library, such as Keras in our case. So, here is a representation of our network with a 224x224 RGB image as input, Xception as first feature extraction plus one dense layer (37 outputs neurons):

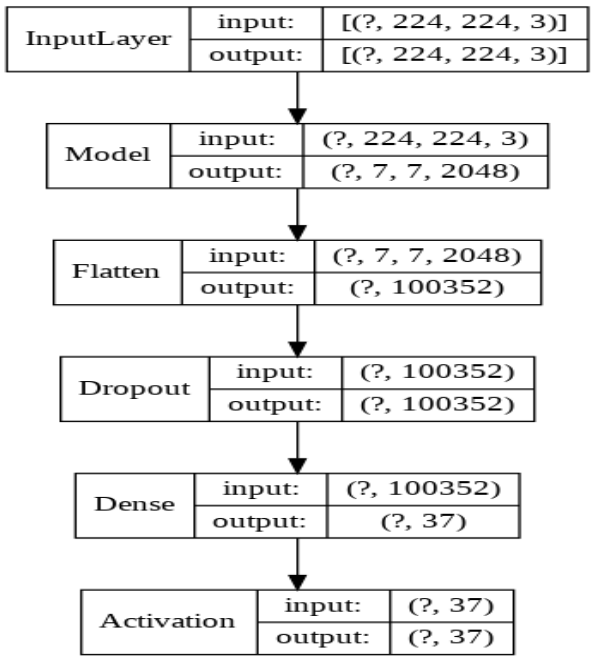


Figure 3. Network layers of Exception(Model layer) plus ours

3.2. Training algorithm

Since we had a regression problem cause of the decision tree that contained the weights (expressed as a percentage from 0 to 1), we found the training samples similar to the query and uses them to classify the new query instance that let the network to be able to predict all the galaxy types by their features. To do this we used the *fit_generator* function provided by Keras that trains the model on data generated batch by batch and which we passed the information of our model to. We compiled the network with the sigmoid activation, the Adam optimizer and the MSE loss function.
The main reason why we used the sigmoid function is because it's a mathematical function that exists between 0 and 1, so it is especially used for models where we have to predict the probability as an output. Since the probability of anything exists only between the range of 0 and 1, sigmoid was the right mathematic choice for us (fig.4).
We chose Adam as optimizer, that is a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement. The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [5].

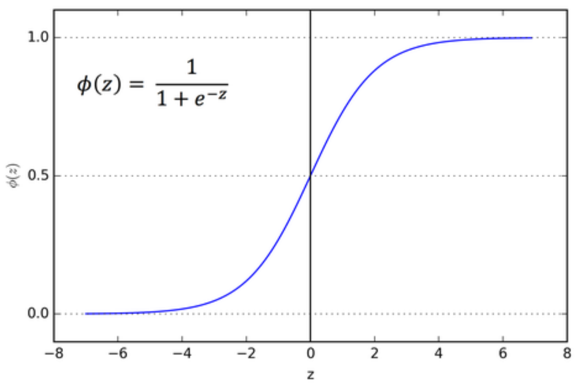


Figure 4. The sigmoid function we used

As regarding the cost function we used the MSE (Mean Squared Error) that uses squared error loss for each training example, known as L2 Loss, i.e. the square of the difference between the actual and the predicted values. It's usually used in the regression technique also because it mathematically represents a positive quadratic function $ax^2 + bx + c, a > 0$ that only has a global minimum, so it is always guaranteed that Gradient Descent will converge to that minimum. In general, the MSE loss function penalizes the model for making large errors by squaring them. This property makes the MSE cost function less robust to outliers.
We also used the RMSE calculated both over training set and validation set to better compare the results with the ones obtained on test set.
We trained our model for 20 epochs with a scheduled Learning Rate (starting from 1e-3) that decreased by 0.1 after 10 epoch steps. The Keras library let us to early stop the training phase if the validation loss had not improved for 4 consecutive epochs.

3.3. Variants and previous attempts

Since we tried to find out the best neural network to perform the task, we had to make a lot of tries before we found the best one. In the very beginning, we started from a Kaggle notebook network, it was quite small and not very performing. After that, we added more convolutional layers to the model above and it started to train better, as we noticed losses on the training set began to decrease.
Yet, networks' performances did not satisfy us at all and so we moved on using some built-in neural network. In this case our choice was Xception because it is a quite new neural network (2017) and it has one of the best trade-offs between accuracy and execution time thanks to the absence of any non-linearity that leads to both faster convergence and better final performance. Starting from Xception, we made some modifications to the original neural network in

order to have 37 output neurons, as much as the classes we have in this scenario, so we found the best model to perform with. In some attempt, we tried to add a batch normalization layer that is used for improving the speed, performance, and stability of artificial neural networks. Batch normalization was introduced in 2015 and it is used to normalize the input layer by adjusting and scaling the activations [4].
Some attempt contained a non-linearity (ReLU) that introduces more degrees of freedom to the model and others contained a final global max-pooling that takes only last dimension of neurons to feed it to the dense part. We tried to use the ReLU because we had a logistic regression to face while we know that deep NNs without non-linearity are essentially a linear regression. In our initial experiments (without Xception) we used also a different loss function, such as the Binary Crossentropy, that has proven to be unsuitable for our regression problem.
All of these attempts are saved in our repository and let us to faster choose the best one.

4. Evaluation

We plotted a graph containing training's losses of the model compared to the validation's ones so that the performance of the network was visible (fig.5). Unlike what we proposed, confusion matrix, that would have been needed for the quantitative analysis, turned out not to be good for our purpose due to the decision tree which does not allow a rapid comparison between the weights of the predictions and the expected values.

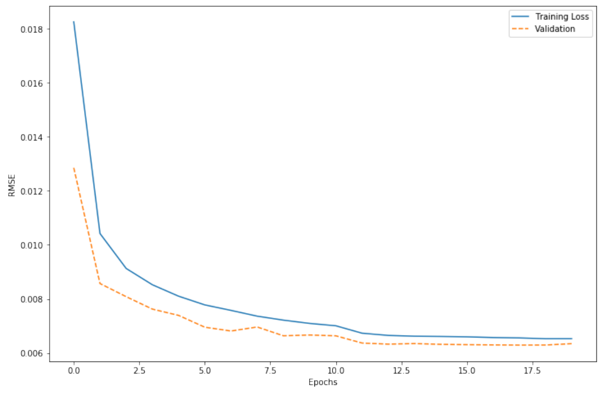


Figure 5. Losses function graph for the best model

So we calculated RMSE:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\widehat{yi} - yi)^2}{n}}$$

based on the prediction of the training set that was previously split into two parts, i.e. it was used 10% of the dataset for validation and 90% for the training set, finding RMSE = 0.07971 and MSE = 0.00628 with the best model we were able to create (RMSE metric is used to make it possible having a performances' comparison between the training and validation set).
This score is worth a 7th place on the competition leaderboard and was quite good, so we would have expected a good score calculating the RMSE on the test dataset. Since we had no weights for test dataset, we used the *central_pixel_benchmarck.csv*, a simple benchmark, provided in the Kaggle competition, that clusters training galaxies according to the color in the center of the image and then assigns the associated probability values to like-colored images in the test set.
In this way, we performed an element-wise subtraction from the prediction and the true (expected) values, then we squared the result, got the average for each row and we executed a square root operation for each row. After that we got the mean error of the rows (fig.7).
Contrary to what we expected, the result (RMSE = 0.133806) was too high and was not good at all, so we thought to two possible issues: there might exist a better implementation of the neural network or the ".csv" provided was not suitable. So we decided to focus on the second issue searching for a new dataset with reliable weights. Since we did not manage to find a dataset good enough for our purpose, we opted to create an acceptable dataset splitting the initial training dataset, that contained the percentages of correct weights, into 80% for train set (nextly split in 90% for train phase and 10% for validation phase) and 20% for test set. We plotted the loss cost function for both training and validation that showed a little but unimportant overfitting (fig.6).

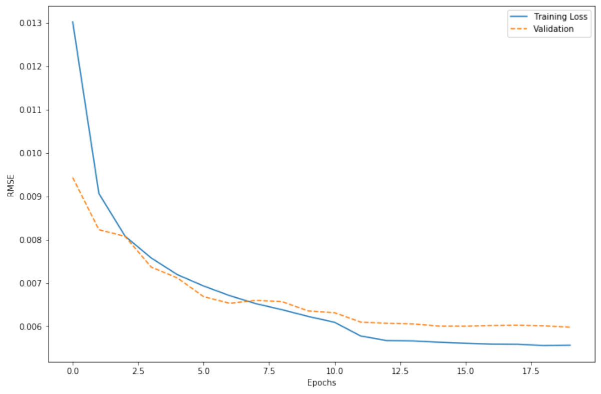
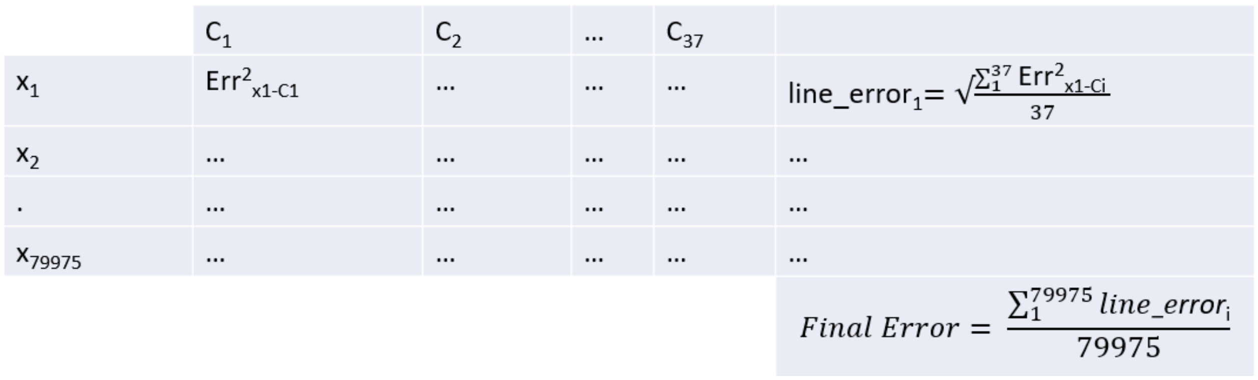


Figure 6. Losses function graph for new training dataset

We obtained a RMSE = 0.0744 during the training so, like before, we would have expected a good score calculating the RMSE on the new test dataset. Effectively, this time, the RMSE on test (computed as the previous image (fig.7) with different predictions' length) is shown to be very close and better than the one over the train dataset, getting a value



	C_1	C_2	...	C_{37}	
x_1	$Err^2_{x1\text{-}C1}$	...	...	...	$line\_error_1 = \sqrt{\frac{\sum_1^{37} Err^2_{x1\text{-}Ci}}{37}}$
x_2	...	...	...	...	...
.	...	...	...	...	...
x_{79975}	...	...	...	...	...
					$Final\ Error = \frac{\sum_1^{79975} line\_error_i}{79975}$

Figure 7. Algorithm to compute RMSE on predictions and test's weights

of **0.07298**. This result scored the 1st position in the leaderboard.
At last, we tried our network for predicting a small dataset of 16 images (added manually), to see qualitatively how it performs, showing the labels associated after the predictions. We chose such small images amount because it comes easier to see fastly the features, and the predictions result very accurate. Here is a table showing predicted labels and the respective images:

GalaxyID	Label
2000.jpg	disc, spiral, tight arms, more than 4 arms, little bulge
2001.jpg	disc, spiral, tight arms, more than 4 arms, little bulge
2002.jpg	disc, spiral, tight arms, 2 arms, obvious bulge
2003.jpg	disc, spiral, tight arms, 2 arms, little bulge, merger
2004.jpg	disc, spiral, tight arms, more than 4 arms, little bulge, odd feature
2005.jpg	disc, spiral, loose arms, more than 4 arms, no bulge, irregular
2006.jpg	disc, spiral, tight arms, cannot count arms, little bulge, odd feature
2007.jpg	disc, spiral, tight arms, cannot count arms, no bulge, odd feature
2008.jpg	disc, spiral, tight arms, cannot count arms, dominant bulge, odd feature
2009.jpg	disc, spiral, tight arms, cannot count arms, no bulge, odd feature
2010.jpg	disc, spiral, tight arms, cannot count arms, little bulge, odd feature
2011.jpg	star or artefact
2012.jpg	disc, spiral, tight arms, more than 4 arms, little bulge
2013.jpg	star or artefact
2014.jpg	star or artefact
2015.jpg	star or artefact

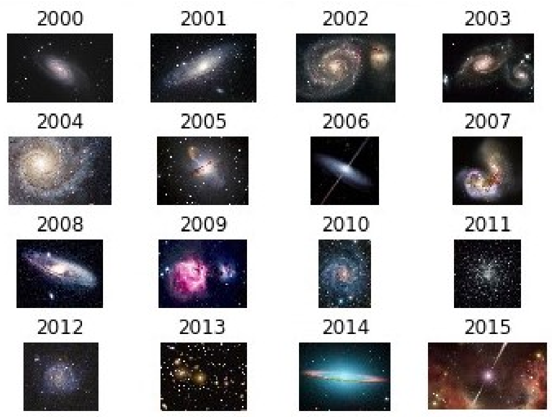


Figure 8. 16 images wth id corresponding to the GalaxyID in the table (there is an outlier in image 2009)

5. Conclusion

If we had more time, we would have tried to manually implement a network, as much as possible, not based on any existing one in order to further optimize the final result. For example, we would have tried to do:

- different or more data augmentation: add more batch samples to feed to the training phase;
- some other deeper network (already existing): more computational time and possible different hardware/software environment needed;
- a mix between two or more of the best deeper CNNs: try multiple combination of different networks probably having a high computational cost but a better performing model and a heavier dimension in byte of the resulting network (e.g. hundreds of millions parameters).

In conclusion, it was really stimulating to face the Galaxy Zoo Challenge, also because we are not so expert on the

field, but we tried to read a lot of documentation and papers to get to the best solution, even if sometimes the various edits we made brought us to worse results. Getting the score as the best one on Kaggle Leaderboard made us really proud, even if technology ran into us! We hope we would have the chance to face some more challenges in the future because it is really exciting.
Here you are the link of our GitHub repository that we used to write this documentation across weeks of work and where it can be found also the code implementation: `https://github.com/riccardogabellone/galaxy_recognition_project`

References

[1] `https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/overview`,.

[2] `https://keras.io`,.

[3] François Chollet et al. Xception deep learning. 2017.

[4] Ioffe et al. Accelerating deep network training by reducing internal covariate shift, 2017. paper on Batch Normalization.

[5] Kingma et al. Adam: A method for stochastic optimization, 2015. paper on Adam optimizer.