Galaxy classification using convolutional networks

Nerea Losada **Iñigo Ortega**

Contents

Abstract

In this document we report our proposal for the application of *Convolutional Neural Networks* to the *Galaxy Zoo 2* dataset. The goal of the project is to design a convolutional network that outputs the probability that a given galaxy image belongs to one of the possible categories, being this a supervised classification problem. We have implemented the classification process using mainly *keras* library, from *tensorflow*. We have also used other libraries, such as *pandas*, *numpy*, *matplot* and some more. In addition, we have preprocessed the images, designed the network architecture and trained it, and finally we have validated the network. At the end we have used the GZ2 decision tree to get the answers for each image.

1 Description of the problem

The task we have to solve is to design a *Convolutional Neural Network* that outputs the probability that a given galaxy image belongs to one of the possible categories. *Convolutional Neural Networks* are ones of the most employed *Deep Neural Network* architectures. They are particularly efficient for computer vision tasks such as image classification, which is actually our task.

Galaxy classification consists of, given an image, predicting the probability that it belongs in a particular galaxy class (generally determined by its morphology).

For that aim, we have the data release for *Galaxy Zoo 2* [1], a citizen science project with more than 16 million morphological classifications of 304,122 galaxies drawn from the Sloan Digital Sky Survey. GZ2 measures morphological features that include bars, bulges, and the shapes of edge-on disks, as well as quantifying the relative strengths of galactic bulges and spiral arms.

However, because of having a huge amount of images to train and test, the dataset was reduced in such a way that only a subset of the total amount of the images are used. This way, we use 13000, which we consider to be sufficient.

The database has the following characteristics:

It consists of 61578 images. In order to classify them we have a *training_solutions.csv* file, in which we have the values of each image for each class. In this case, the classes are the possible answers to some questions: there are 11 questions with their possible answers, being each of those the features. This way, we have 37 attributes which their values.

One of the characteristics of this problem is that it is a hierarchical classification [2] problem because of classes being defined as some answers of a questionnaire, giving the classes set taste of a decision tree.

2 Description of our approach

We organized the implementation of the project according to the tasks:

1. Preprocessing the images
2. Designing the network architecture and training it.
3. Designing the validation method to evaluate the score of the *Convolutional Neural Network*

To complete our tasks, we follow some steps that are implemented in the notebook and explained in this report:

1. Understanding the dataset
2. Reading the dataset
3. Preprocessing the data: reducing the amount of images and resizing them
4. Creating train, validation and test sets
5. Designing the network
6. Creating the model and training it
7. Validating the model
8. Visualizing the results
9. Applying the GZ2 decision tree

2.1 Reading the data

We import into *Python* the training set's images from *images_training_rev*1 and their labels from the *training_solutions_rev*1.*csv CSV* file, both provided at *Kaggle* on the *Galaxy Zoo Challenge*. We use the *pandas* library to read the data from the *CSV* file. To do so, we use the *read_csv* function, which automatically gets all the information.

2.2 Preprocessing

The "Galaxy Zoo" dataset consists of a total 141553 images. These are split into 61578 images for training –each with their respective probability distributions for the classifications for each of the inputs– and 79975 images for testing. As a crowd-sourced volunteer effort, images of the dataset were classified across 11 different categories. Each of categories have attributes which volunteers can rank, there are 37 attributes in total.

The votes on these volunteer categorizations are normalized to a floating point number between 0 and 1 inclusive. A number close to 1 indicates many users identified this category for the galaxy image with a high level of confidence, while numbers close to 0 indicate otherwise.These numbers represent the overall morphology of a galaxy in 37 attributes.

2.2.1 Reducing the amount of images

As we have mentioned, there are lots of images, so we reduce our dataset by working with a subset of them. In order to do that, we take the first 13000 images. 10000 would be enough, nevertheless we add 3000 more images for testing.

2.2.2 Resizing the images

All images in the dataset are of size 424×424 and the object of interest is always centered. In order to reduce the dimensionality of the images, during preprocessing images are cropped to 150×150. Cropping can help the *CNN* learn which regions are related to each specific expression as well as improve performance when training [3]. Reducing the size of each image has the secondary quality of being more memory efficient which is beneficial for training and network size.

In order to do that, we read the images from our reduced dataset and we resize them by using the *Python Imaging Library (PIL)*. We save the resized images in an array called x, and the values of each image in an array called y.

2.3 Splitting the data set

We split the dataset in order to have three sets: one for training, another one for validating the model, and the last one for testing. In order to do that, we take the first 10000 images, and we sample them in the following way: %70 for training and the remaining for validating. The last 3000 images are used for testing.

2.4 Designing the network and training

A standard neural network (*NN*) consists of many simple, connected processors called neurons. Each neuron produces a sequence of real-valued activations. A neuron can be activated from an input or another neuron's activation through its weighted connections from a previous layer [4].

An *activation function* computes a weighted sum of its input, adds a bias and decides whether the neuron's value should be propagated or not.

A *loss function* defines the difference between the target and actual output. An *NN* uses the loss function to correct weights after a feed forward operation, this process of correction is called back propagation. The error is propagated backwards from the output layer through the hidden layers to the input layer, wherein the weights and biases are modified in such a way that the error for the most recent input is minimized. Over many training samples the loss function will minimize error and the value of the weights for the whole network will begin to converge. The curve which the error rate of the network experiences is controlled through a gradient descent method. This method can help determine whether the network should be trained further or to stop early.

A *CNN* takes an image as an input and feeds it through several layers; usually convolutional layers with activation functions, pooling layers, and a fully-connected layer (*FCL*). Convolutions are the primary operation of a *CNN* and what makes them distinct from other type of networks. A convolutional layer parameters are made up of a set of small learnable weights known as filters. A filter has a local receptive field, and given an image as an input to a convolutional layer, it convolves each filter across the image's width and height using a specified stride size and produces outputs called feature maps. Filters are what allow a *CNN* to learn to extract visual clues from its input such as edges, lines, and corners [5].

Once the architecture of a *CNN* has been specified, its filters and weights are initialized to small random values. Next, given an image as an input, it is fed through the convolutional, pooling operations, and the *FCL*. The output probabilities of the network are then used to compute the total error, and finally gradient descent is used to update the filter and weight values with respect to their contribution to the total error. This process is repeated with other images in the dataset until satisfactory results are achieved.

In order to create our model, we first need to specify its architecture.

2.4.1 Network architecture

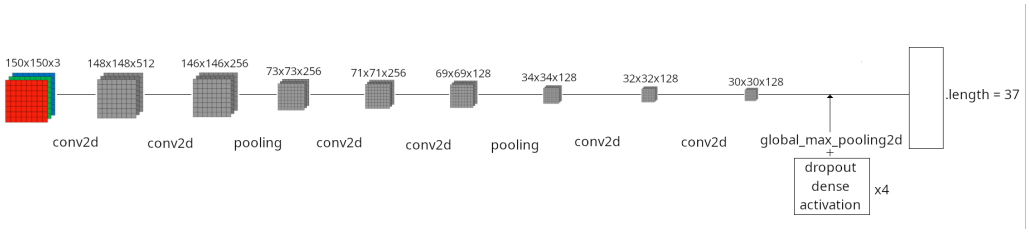

Figure 1: Our network architecture.

As it is shown in [Figure 1], we have used 6 Convolutional and 3 Pooling layers. Afterwards, we have used four times almost the same architecture of layers: a dropout, a fully connected layer and its activation.

Sequentially explained, the following is the way the network has been designed for an input of shape $150 \times 150 \times 3$ (a RGB image):

a) 3×3 Convolutional hidden layer, outputting 148×148 sized 512 channels.

b) 3×3 Convolutional hidden layer with *ReLU* activation, outputting 146×146 sized 256 channels.

c) 2×2 Max Pooling hidden layer, outputting 73×73 sized 256 channels.

d) 3×3 Convolutional layer, outputting 71×71 sized 256 channels.

e) 3×3 Convolutional hidden layer with *ReLU* activation, outputting 69×69 sized 128 channels.

f) 2×2 Max Pooling hidden layer, outputting 34×34 sized 128 channels.

g) 3×3 Convolutional hidden layer, outputting 32×32 sized 128 channels.

h) 3×3 Convolutional hidden layer with *ReLU* activation, outputting 30×30 sized 128 channels.

i) A Global Max Pooling layer, outputting 128 channels with a single value on each. So they are basically 128 values.

j) Dropout layer with 0.25 probability of dropping, outputting 128 values.

k) A fully connected layer with *ReLU* activation, outputting 128 values.

l) Dropout layer with 0.25 probability of dropping, outputting 128 values.

m) A fully connected layer with *ReLU* activation, outputting 128 values.

n) Dropout layer with 0.25 probability of dropping, outputting 128 values.

o) A fully connected layer with *ReLU* activation, outputting 128 values.

p) Dropout layer with 0.25 probability of dropping, outputting 128 values.

q) A fully connected layer with *Sigmoid* activation, outputting 37 values.

Convolutional layer [6]

A convolutional layer contains a set of filters whose parameters need to be learned. The height and weight of the filters are smaller than those of the input volume. Each filter is convolved with the input volume to compute an activation map made of neurons. The output volume of the convolutional layer is obtained by stacking the activation maps of all filters along the depth dimension. Since the width and height of each filter is designed to be smaller than the input, each neuron in the activation map is only connected to a small local region of the input volume. In other words, the receptive field size of each neuron is small, and is equal to the filter size.

In addition, as the activation map is obtained by performing convolution between the filter and the input, the filter parameters are shared for all local positions. The weight sharing reduces the number of parameters for efficiency of expression, efficiency of learning, and good generalization.

Pooling layer [7]

After a convolution layer, it is common to add a pooling layer in between *CNN* layers. The function of pooling is to continuously reduce the dimensionality to reduce the number of parameters and computation in the network. This shortens the training time and controls overfitting.

The most frequent type of pooling is max pooling, which takes the maximum value in each window. This decreases the feature map size while at the same time keeping the significant information.

Fully-connected layer [8]

The result of the *Convolutional layer* and the *Max-pooling layer* feeds into a fully connected neural network structure that drives the final classification decision.

Dropout [9]

Dropout is a method to prevent *NN*'s from overfitting. It does this by randomly deactivating some neurons in one –or many– layers. This makes the remaining active neurons compensate their weights twice as much in the back propagation phase. Neurons are randomly activated and deactivated per update or depending on a probability value, for example, in our case we have always used a dropout ratio of 0.25. By activating and deactivating neurons in layers, dropout keeps the neurons of a network robust to the training data by never letting them fully fit to the training data.

Activation function [10]

Activation function is just a thing function that is used to get the output of node. It is also known as Transfer Function. It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

In this case, we have used the *ReLU* activation function for every layer except the last one. At the begging we thought about using a *softmax* layer as the last one, but then we realised that the values predicted are not the probability of belonging to each class, so the sum of the values of all features is not 1. Therefore, for the last layer we have used the *sigmoid function*.

2.5 Validation

In order to validate the network, we kept 3000 images away from the training set (besides the 3000 from the test set).

With 7000 images for training and 3000 for validation, we trained the model and got the result showed in [Figure 2].

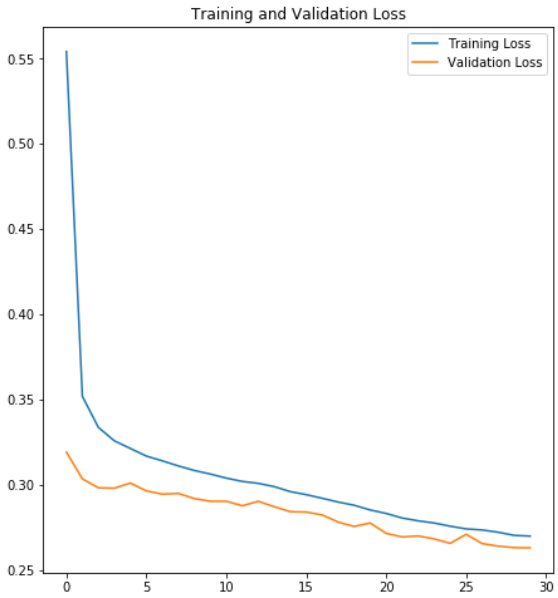

Figure 2: Training and Validation loss change through the execution of training

The figure shows how the validation set loss is even less than the one of the training set. That means the model is able to generalize properly.

2.6 GZ2 Decision Tree

The [Figure 3] shows the decision tree used to get the answers to each task for each image. The *GZ2* tree has 11 classification tasks with a total of 37 possible responses. A classifier selects only one response for each task, after which they are immediately taken to the next task in the tree. Tasks 01 and 06 are the only questions that are always answered for each and every classification.

Task	Question	Responses	Next
01	*Is the galaxy simply smooth and rounded, with no sign of a disk?*	smooth	07
		features or disk	02
		star or artifact	**end**
02	*Could this be a disk viewed edge-on?*	yes	09
		no	03
03	*Is there a sign of a bar feature through the centre of the galaxy?*	yes	04
		no	04
04	*Is there any sign of a spiral arm pattern?*	yes	10
		no	05
05	*How prominent is the central bulge, compared with the rest of the galaxy?*	no bulge	06
		just noticeable	06
		obvious	06
		dominant	06
06	*Is there anything odd?*	yes	08
		no	**end**
07	*How rounded is it?*	completely round	06
		in between	06
		cigar-shaped	06
08	*Is the odd feature a ring, or is the galaxy disturbed or irregular?*	ring	**end**
		lens or arc	**end**
		disturbed	**end**
		irregular	**end**
		other	**end**
		merger	**end**
		dust lane	**end**
09	*Does the galaxy have a bulge at its centre? If so, what shape?*	rounded	06
		boxy	06
		no bulge	06
10	*How tightly wound do the spiral arms appear?*	tight	11
		medium	11
		loose	11
11	*How many spiral arms are there?*	1	05
		2	05
		3	05
		4	05
		more than four	05
		can't tell	05

Figure 3: The GZ2 decision tree, comprising 11 tasks and 37 responses. The 'Task' number is only an abbreviation and does not necessarily represent the order of the task within the decision tree. The texts in 'Question' and 'Responses' are displayed to volunteers during classification. 'Next' gives the subsequent task for the chosen response.

In order to get the answer to each question for each image, we have implemented a function that passing the values of an image, returns the corresponding answer path, this is, its behaviour follows the steps in [Figure 3], building a path of answers.

In [Figure 4] we can see that behaviour visually.

Example of answer paths are: ['smooth', 'no'] and ['features or disk', 'no', 'yes', 'yes', 'loose', '2', 'just noticeable', 'no'].

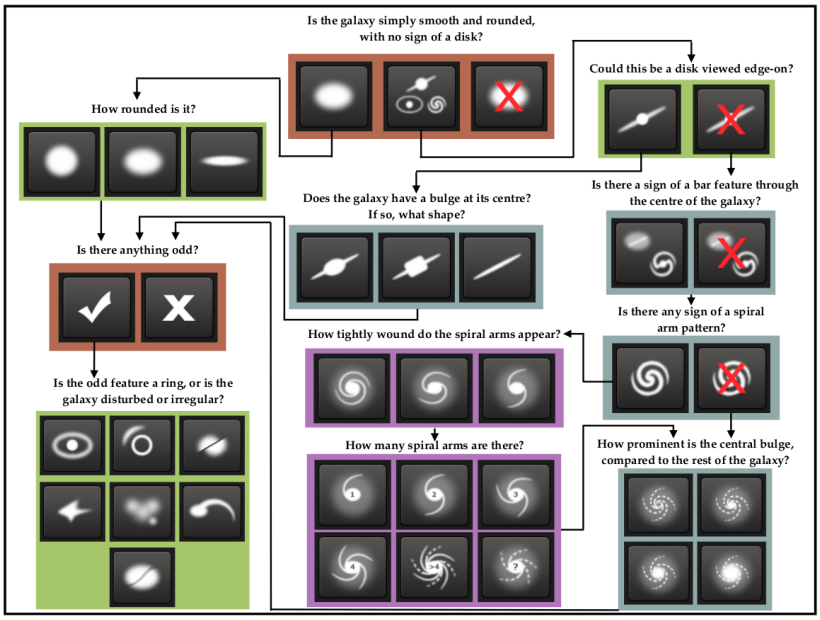

Figure 4: Flowchart of the classification tasks for the "Galaxy Zoo" dataset. Tasks are color-coded by their relative depths in the decision tree. Tasks outlined in brown are asked for every galaxy. Tasks outlined in green, blue, and purple are one, two or three steps respectively below branching points in the decision tree [Figure 3].

But in order to get the mentioned paths, we first take the test set and predict the values for each image. Afterwards, making use of the [Figure 3] table, we convert each of the predictions made on the images, getting a list of answer paths. One path for each image.

However, the table does not talk about probability values on each question, so we consider the answer for each question the one with the highest probability.

Then, we do the same using the real values that we have in the *csv* file that contains the solutions. This way we will have another set of paths with the predicted answers for each image and another one with the true answers.

Once we have those both the answer paths generated from the model applied to the images and the ones obtained from the labels, we can compare them to know how many images has the model predicted correctly.

3 Implementation

All the project steps were implemented in Python. We mainly use the following libraries for creating the *CNN* and training the model:

- ***pandas***: In order to retrive the data from the *csv* files provided by the team responsible of *Galaxy Zoo Challenge* at *Kaggle*.

- ***tensorflow.keras***: On this notebook *Keras* is used, especifically, the version bundled with *TensorFlow*.
- ***keras_preprocessing***: It is used for image manipulation, for preprocessing or demonstration porpouses.
- ***os, random, shutil***: They are used at the moment of reading files from the system into python.

Some other libraries are also used, such as *matplotlib*, *numpy* or *PIL*. We illustrate how the implementation works in the Python notebook *Galaxy Zoo CNN MLNN.ipynb*.

4 Results

Applying the model to the test set, getting the answer paths from the generated values and comparing them with the paths outputted from the true labels, we get the following accuracy: %41. Out of 3000 images, 1229 images have been labelled correctly.

It has to be mentioned, that this accuracy is calculated comparing 2 answer paths completely. That is, it doesn't matter how similar 2 paths are, if there is a single difference, they are considered unequal.

5 Conclusions

The Galaxy Zoo Challenge was, unquestionably, interesting and even entertaining, however, because of it being a hierarchical classification problem, it caused a lot of struggles at the time of designing the model and understanding how the classification should eventually be made.

Regarding the results, %41 percent accuracy is, of course, not the best that we could achieve, but the best we were able to. However, it is pretty good bearing in mind that %41 of the answer paths were exactly the same as their corresponding labels and not just similar.

References

[1] "Galaxy zoo 2 challenge page." `https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge`.

[2] "Hierarchical classification." `https://www.kdnuggets.com/2018/03/hierarchical-classification.html`.

[3] M. Ghayoumi, "A quick review of deep learning in facial expression," *Journal of Communication and Computer*, vol. 14, p. 34–38, 2017.

[4] J. Schmidhub, "Deep learning in neural networks: An overview," *the official journal of the International Neural Network Society*, vol. 61, 2015.

[5] L. B. Yann Lecun, Patrick Haffner and Y. Bengib, "Object recognition with gradient-based learning," *In Contour and Grouping in Computer Vision*, Springer, 1999.

[6] "Convolutional layer page." `https://www.sciencedirect.com/topics/engineering/convolutional-layer`.

[7] "Pooling layer page." `http://cs231n.github.io/convolutional-networks`.

[8] "Fully connected layer page." `https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks`.

[9] R. S. N. S. G. H. Alex Krizhevsky, Ilya Sutskever, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[10] "Activation function page." `https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6`.

[11] I. Ortega and N. Losada, "Github repository." `https://github.com/initega/galaxy-zoo-cnn-mlnn`.

GitHub: Our development was carried through the cited git repository [11].