Galaxy classication using convolutional networks

Xabier Lekunberri and Iker Ortiz

Contents

Abstract

In this document we report our proposal for the application of the techniques of neural networks to galaxy classification problem. The neural network that we have implemented is a Convolutional Neural Network. We have used some libraries such as *Keras* (a front-end of *TensorFlow*), *numpy* and *pandas*. We have learned a model from a fraction of the training data of The Galaxy Challenge[1] from *Kaggle*.

1 Description of the problem

The task we have to solve in this project is to get a solution for classification problem using a CNN, using the data from *Galaxy Zoo*[2], which is astronomy project where Internet users are invited to answer simple questions about what can be seen in an image of a galaxy to assist in its classification. The questions are about the shape of the galaxy shape and some other features that a person can easily determine. Following the flowchart of Figure 1, we can correctly classify any galaxy. After finishing to answer the questions, these answers are aggregated to a database and converted into probabilities.

This is where a deep learning approach like a CNN could be very useful, where instead of requiring human help in order to classify by hand, this assignment could be done automatically and the probability of each answer calculated via regression.

2 Approach

2.1 Preprocessing

The original resolution of the images is 424x424 and 3 channels for color. As this is a lot of data to process, we decided to resize the images to a size of 212x212; this is done with the auxiliary function `resize_image()`. This function takes a random section of 212x212 from the original image and it can randomly be flipped horizontally. Also, as we only have the classes for the train images, we will split it in train and test sets. We decided to use 50% of the images for train and the other 50% for test.

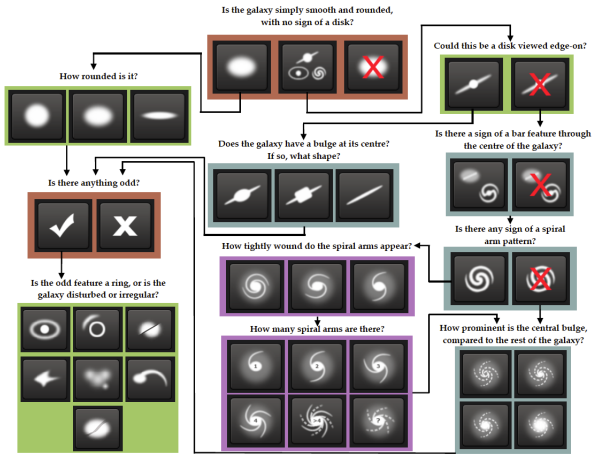

Figure 1: Flowchart of the classification for *Galaxy Zoo*

[1] Galaxy Zoo - The Galaxy Challenge: `https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/data`
[2] `https://www.galaxyzoo.org/`

2.2 Network Architecture

The architecture of the network that we decided to implement is a mixture of many of the references that we checked. Even so, most of the architectures followed a similar pattern: Some convolutional layers first and before outputting, there were some fully connected or dense layers (but fewer than convolutional ones). The last one always outputs a 37-units array, the number of probabilities shown at the `training_solutions_rev1.zip` file at the *Kaggle* site[1].

On the one hand, we chose to use 6 convolutional layers and 3 pooling layers, as it seemed to be the average; in the other hand, we decided to adhere 3 fully connected layers at the end of the network, for the same reason. It should be mentioned that we added a *flatten* layer before the last dense layer, which unrolls the input values to adapt to the last layer, because during the network.

With regard of the activation function, we used ReLu instead of Sigmoid or tanh, due to the better results of the first one. To avoid overfitting, we applied *dropout* to the first 2 dense layers, with a drop rate of 0.5. Moreover, a table with the whole architecture of the network is shown at Table 1.

2.3 Implementation

The material from *Galaxy Zoo*[1] challenge consist of more than 61000 photos and a `.csv` file with the name of each photo and the probability of being of each of the 37 classes. Using *numpy arrays* and *pandas* to read the `.csv` file and create a *dataframe*, we load the photos and the solutions to different *arrays* (e.g. `x[i]` contains a photo and `y[i]` contains the 37 probabilities).

Once all the information is loaded (took 5 minutes more or less to load in our computer) the train and test sets are generated using `train_test_split()` method from *scikit_learn*'s *model_selection* library. Doing this we get two sets of 30500 photos each one.

Next, we have the option to create and train a new model or load a previously trained one; both options will use the *Keras* library. With this project we provide a model with this characteristics:

- test_size = 0.5
- epochs = 10
- batch_size = 32

If we decide not to load a previously trained model, the `create_model()` function will create the network defined at Table 1 using the *Keras* library and the model is going to be trained using the loss function of *Mean Square error* and the *Stochastic Gradient Descent* optimizer with learning rate as 0.1. It took about 1 hour to train this model using a computer with these characteristics:

- CPU: Intel® Core™ i7 4790K
- GPU: NVIDIA GeForce GTX 1080
- RAM: 16 GB

Layer	1	2	3	4	5	6	7	8	9
Stage	conv + max	conv + max	conv	conv	conv	conv + max	full + drop	full + drop	full
No. filters	48	96	192	192	384	384	-	-	-
Out. Neurons	-	-	-	-	-	-	2048	2048	37
Filter size	5x5	5x5	3x3	3x3	3x3	3x3	-	-	-
Pooling size	3x3	2x2	-	-	-	3x3	-	-	-
Pooling stride	3x3	2x2	-	-	-	3x3	-	-	-
Activation	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU	ReLU

Table 1: Network Architecture

2.3.1 Overfitting

So as to avoid overfitting, we tackled this problem with 2 approaches:

- Rather than just cropping or resizing the images, we did a little data augmentation by cropping from the center of the image, but from a random location; as well as flipping horizontally with random probability.
- Another method to avoid overfitting is the dropout. Dropout reduces the overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks.

3 Results and conclusions

We use the accuracy to measure our models, reaching a maximum of 50%. We understand that this is not a very good result, but as it takes more than one hour for each iteration (change the parameters, train the model and test it) we had not enough time to do many tests.

Moreover, surely there are much better network structures, but as said, due to the lack of time we could not test any apart from the one in Table 1. Some of the improvements that we could do in the future could be tuning the parameters of the convolutional and/or pooling layers or the quantity of them itself, which varies depending on the project, and in our case an architecture that had worse perfomance than others could fit in our implementation and enhance our results.

4 References

[1] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jrgen Schmidhuber. Flexible, high performance convolutional neural networks for image classication. In *IJCAI Proceedings-International Joint Conference on Articial Intelligence*, volume 22, page 1237. Barcelona, Spain, 2011.

[2] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675 678. ACM, 2014.

[3] Yoon Kim. *Convolutional neural networks for sentence classication*. CoRR, abs/1408.5882, 2014.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classication with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 10971105, 2012.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[6] Jorge De La Calleja and Olac Fuentes. Machine learning and image analysis for morphological galaxy classication. *Monthly Notices of the Royal Astronomical Society*, 349(1):8793, 2004.

[7] Kyle W Willett, Chris J Lintott, Steven P Bamford, Karen L Masters, Brooke D Simmons, Kevin RV Casteels, Edward M Edmondson, Lucy F Fortson, Sugata Kaviraj, William C Keel, et al. Galaxy Zoo 2: detailed morphological classications for 304 122 galaxies from the Sloan Digital Sky Survey. *Monthly Notices of the Royal Astronomical Society*, 435(4):28352860, 2013.

[8] AbdulWahab Kabani, and Mahmoud R. El-Sakka. *How Important is Scale in Galaxy Image Classication?*.

[9] Ji Wan, Dayong Wang, Steven C.H.HOI, Pengcheng Wu, and Jianke Zhu. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the ACM International Conference on Multimedia*, pages 157 166. MM, 2014.