# Galaxy classification using convolutional networks

**Ainhoa Serrano**

## Abstract

In this document I report my proposal for the application of CNNs for the prediction of the probability of a galaxy belonging to some classes. I have implemented the process using the pandas and keras libraries. The models have learned Convolutional Neural Networks using the train data and computing the root mean square error in the validation data for each epoch. The final testing was made on a separate test set. From the results, the best model's evaluation error was 0.1161.

## Contents

# 1 Description of the problem

The task to be solved is the galaxy classification, which consists of given an image, predicting the probability that it belongs in a particular galaxy class (generally determined by its morphology). Galaxies can be of different shapes, colors and sizes: for example they can take the shape of spirals.

In order to solve this problem, this is the given data distributed in the following files and folders:

- **images_training**: folder with JPG images of 61578 galaxies. Files are named according to their GalaxyId.
- **training_solutions_rev1**: CSV file with the probability distributions for the classifications for each of the training images.

The CSV file's first column indicates the galaxy IDs. The following 37 columns are all floating point numbers between 0 and 1 inclusive, representing the probability of each galaxy belonging to the class of the column. More specifically, these represent the morphology (or shape) of the galaxy in 37 different categories as identified by crowdsourced volunteer classifications as part of the Galaxy Zoo 2 project. These morphologies are related to probabilities for each category; a high number (close to 1) indicates that many users identified this morphology category for the galaxy with a high level of confidence. Low numbers for a category (close to 0) indicate the feature is likely not present.

For the development of this project, the following references were considered: [1], [2], [3].

# 2 Description of my approach

The project implementation was organized according to the tasks:

1. Organization of the data.
2. Preprocessing of the images.
3. Design of the network architecture and training.
4. Validation.

## 2.1 Organization of the data

Firstly, it it important to mention that the given dataset has 61578 instances, but I decided to use 10000 instances to train the data, and 2000 more to test it. I decided to use less instances and then apply data augmentation to check if good results could be achieved with only 10000 instances of data. This is because collecting data can be a really demanding task, so it would be helpful if the data augmentation could replace the collection of data.

In any case, the computational load was too much for my computer, so instead of creating and training the models in a local path, I have decided best to use *Google Colab*. To make use of it, I have had to upload the 12000 images to *Google Drive* in folders of 1000 images, so that the uploads don't crash. This is shown in the first cells of the jupyter notebook.

The data was split into 3 folders:

1. Train: 8000 images to do the training.
2. Validation: 2000 images to validate the network.
3. Test: 2000 images to test the network.

The notebook is located in the same folder as these three folders.

## 2.2 Preprocessing

I used the *pandas* library to load and represent the dataset as a dataframe. The dataset has the same structure as the .csv file with the values of the classes, so in the first column of the dataset the IDs of

the galaxies are saved and the rest of the columns represent the probability of each galaxy belonging to the class of the column.

After loading the dataset into a dataframe, I splitted the dataframe into 3 smaller ones: one for the training data, a second one for the validation data and a third one for the test data. This was done because the keras' generators of data need the data to be in different folders and dataframes.

As commented before, the next step was to create the generators of data. These generators generate batches of tensor image data that the model can use as the data will be looped over (in batches). They also can be used to create augmented data. In this case, I applied several types of augmentations. These were the parameters introduced in the generators to create augmented data:

1. **rescale=1./255** : this parameter scales the data so that its values are contained between 0 and 1.
2. **rotation_range=40** : Degree range for random rotations
3. **width_shift_range=0.2** : It shifts the image to the left or right(horizontal shifts).
4. **height_shift_range=0.2** : It shifts the image up or down (vertical shifts).
5. **shear_range=0.2** : distortion of the image along an axis, to create or rectify the perception angles.
6. **zoom_range=0.2** : Range for random zoom
7. **horizontal_flip=True** : Randomly flip inputs horizontally.

It should be pointed out that the augmentation of the data was only applied to the training generator and not to the validation generator, because we don't need more data to test the model, we just need it for the training set as the augmentation helps it to generalize better.

In the following figure we can see two images of the original data, that is, not augmented data:

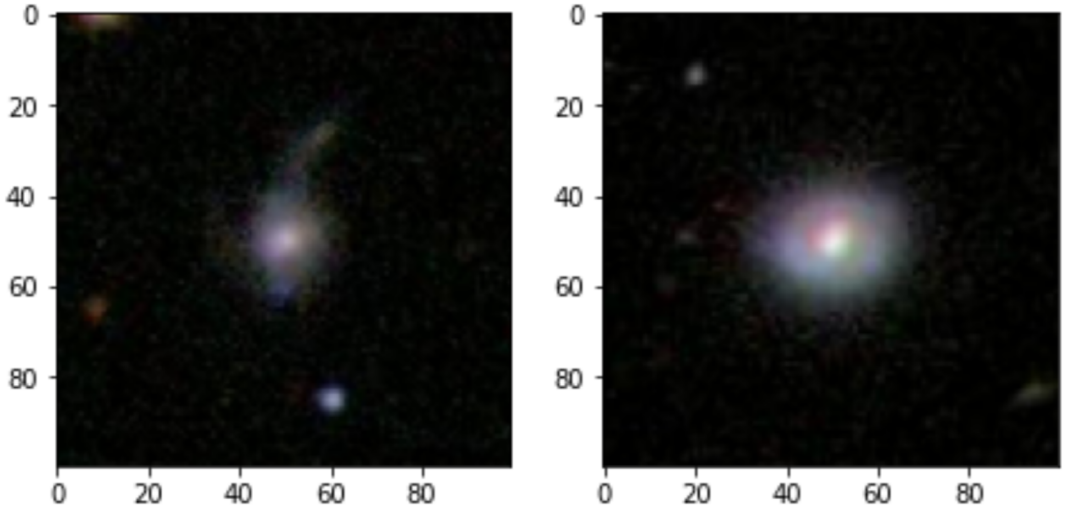



Figure 1: Not augmented data

On the other hand, in the following 2 images of the augmented data are shown. As it can be seen, the images are somewhat distorted, which can be appreciated mostly in the borders.

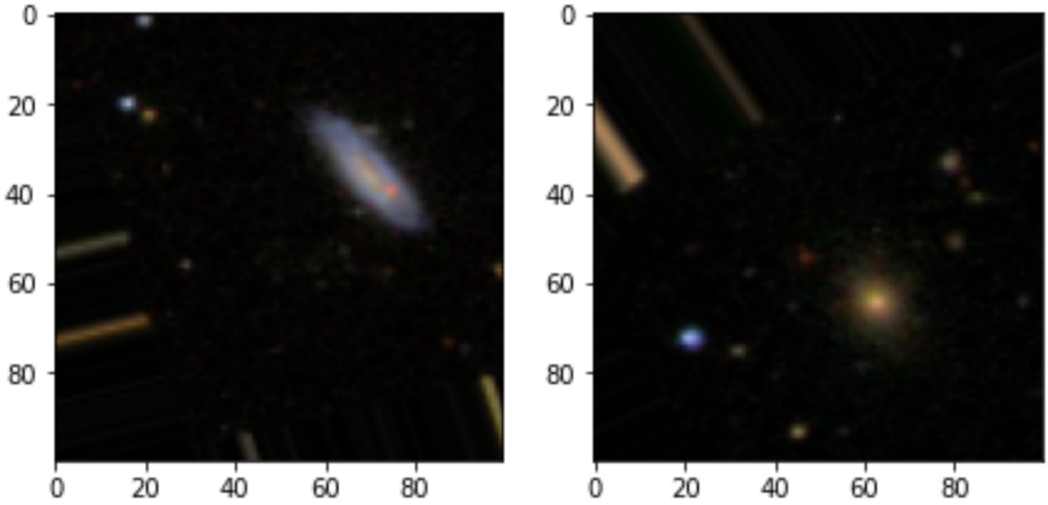



Figure 2: Augmented data

Apart from this, I realized that the images contained a lot of useles data, because the region of interest in each image is the galaxy, but the images show a lot of black space, which is worthless for the training. For this, I cropped the image taking the center, but not so much to risk cropping the galaxy. The cropping size was from (424,424) to (250,250). This are some of the obtained images using cropping:

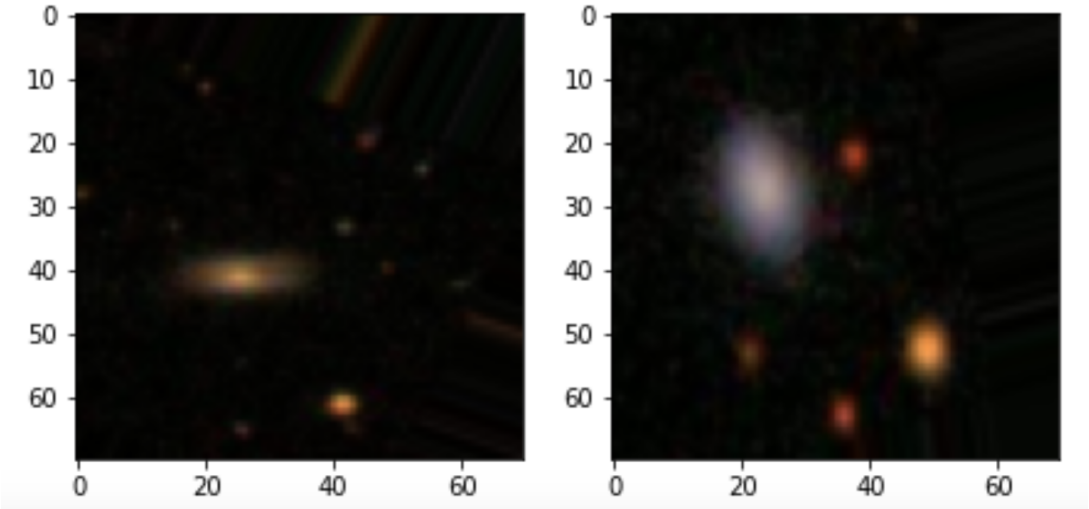



Figure 3: Cropped data

### 2.3 Models

In this section, I will explain the models I created and trained. These are the characteristics:

1. First approach:
   (a) **Structure**:
   - Convolution with 3 channels in and 32 filters. ReLU activation.
   - 2x2 Max-pooling
   - Convolution with 3 channels in and 64 filters. ReLU activation.

- 2x2 Max-pooling
- Convolution with 3 channels in and 128 filters. ReLU activation.
- 2x2 Max-pooling
- Convolution with 3 channels in and 128 filters. ReLU activation.
- 2x2 Max-pooling
- Flatten the input
- Dropout of 0.5
- Fully connected layer with 512 output units. ReLU activation.
- Output layer with 37 units. Sigmoid activation.

(b) **Input size**: (100,100). The images were reescaled from (424,424) to (100,100) to reduce the training time.

(c) **Batch size**: 100

(d) **Loss function** : mean square error

(e) **Optimizer** : adamax

(f) **Metric** : root mean square error

2. Second approach

(a) Structure:

- Convolution with 3 channels in and 32 filters. ReLU activation.
- 2x2 Max-pooling
- Convolution with 3 channels in and 64 filters. ReLU activation.
- 2x2 Max-pooling
- Convolution with 3 channels in and 128 filters. ReLU activation.
- 2x2 Max-pooling
- Convolution with 3 channels in and 128 filters. ReLU activation.
- 2x2 Max-pooling
- Flatten the input
- Dropout of 0.5
- Fully connected layer with 512 output units. ReLU activation.
- Output layer with 37 units. Sigmoid activation.

(b) **Input size:** (70,70). The images were first cropped from (424,424) to (250,250) and then resized to (70,70). This was done because the images contained a lot of black space which is irrevelevant data, so part of that black space is cropped.

(c) **Loss function** : mean square error

(d) **Optimizer** : adamax

(e) **Metric** : root mean square error

3. Third approach

(a) Structure:

- Convolution with 3 channels in and 32 filters. ReLU activation.
- 2x2 Max-pooling
- Convolution with 3 channels in and 96 filters. ReLU activation.
- 2x2 Max-pooling
- Convolution with 3 channels in and 128 filters. ReLU activation.
- 2x2 Max-pooling
- Convolution with 3 channels in and 128 filters. ReLU activation.
- 2x2 Max-pooling
- Flatten the input
- Dropout of 0.5
- Fully connected layer with 96 output units. ReLU activation.
- Dropout of 0.5
- Fully connected layer with 64 output units. ReLU activation.

- Output layer with 37 units. Sigmoid activation.

(b) **Input size:** (70,70). The images were first cropped from (424,424) to (250,250) and then resized to (70,70).

(c) **Loss function** : mean square error

(d) **Optimizer** : adamax

(e) **Metric** : root mean square error

4. Fourth approach: ResNet

(a) This last one is a pre-trained model trained on the *imagenet* dataset. For this one, I have added some layer in order to adapt it to this problem.

(b) Added structure:

- GlobalAveragePooling2D
- Flatten the output
- Fully connected layer with 512 output units. ReLU activation.
- Dropout of 0.5
- Output layer with 37 units. Sigmoid activation.

(c) **Input size:** (70,70). The images were first cropped from (424,424) to (250,250) and then resized to (70,70).

(d) **Loss function** : mean square error

(e) **Optimizer** : adamax

(f) **Metric** : root mean square error

### 2.4 Validation

In order to validate the models, while training, the models calculated the prediction error of the training data on each epoch as well testing it with the validation images, which is a separate set of data. The evaluation metric used for this task was the root mean square error, defined as follows:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2}$$

Where:

$N$ is the number of galaxies times the total number of responses(each probability value). So in this case $N$ = No. Classes * No. Galaxies
$pi$ is the predicted value
$ai$ is the actual value

Aditionally, I also used the models to predict the target values for the test set, in order to check if the evaluation that the model makes with the validation data while training is correct.

## 3 Implementation

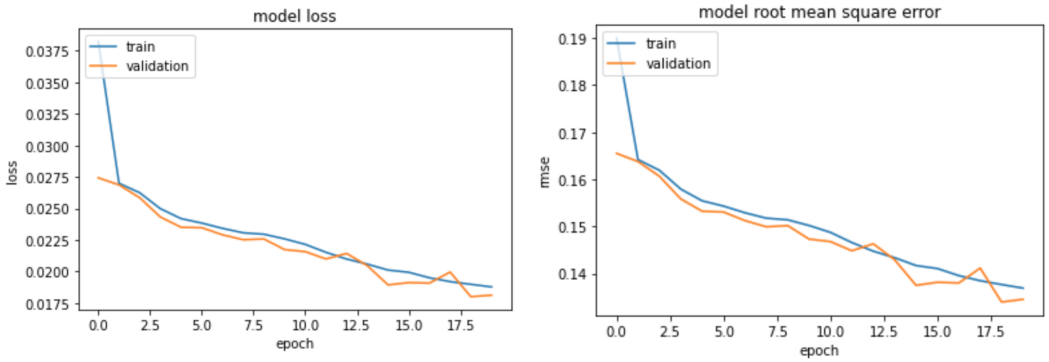
All the project steps were implemented in Python. I used pandas library for reading the target values of the file *training_solutions_rev1*, and Keras for reading the images, the model training and testing. I illustrate how the implementation works in the Python notebook `P50_Galaxy_Conv_DL_D2`.

## 4 Results

In the following section, I am going to describe each model's performance in the training time for the validation data, and for the test data.

### 4.1 First approach

This model used augmented data but not the cropping. The number of epochs for the training was 20. The history of the training time outputs the following graph:




(a) Loss evolution

(b) RMSE evolution

Figure 4: Loss and RMSE for the first model

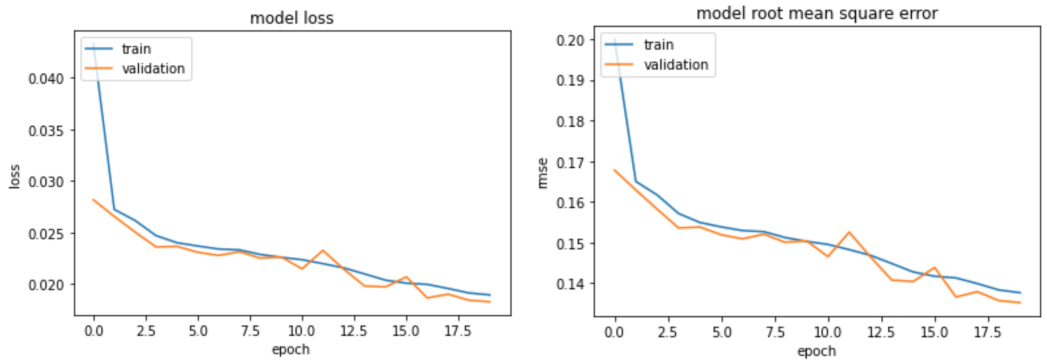
Taking a look into the graphs, we can see that the loss and the error start with a high value, but they are rectified in the following epochs, reaching a value of 0.1345 for the error of the validation data in the last epoch.

The difference between the validation data error and the training data error is minimal, as the latter's value is 0.1330, so it is a good sign that the model is not overfitting.

For further assesment, I used the test data to predict its values and compare it to the actual values. In this case, the prediction gave an error of 0.1364, a very similar value to the validation data, so the test prediction is consistent with the performance of the model in the training.

### 4.2 Second approach

This second model used the augmented data using the cropping method previously explained. However, the model structure is the same as the previous one, with the purpose of comparing the effect it had cropping the images. The history of the training time outputs the following graphs for the loss and the error evolution:




(a) Loss evolution

(b) RMSE evolution

Figure 5: Loss and RMSE for the second model

As it can be seen, the model's traning loss started with a value of 0.043 aproximately, and after 20 epochs, it was reduced to 0.019. The validation data values' evolution was not as smooth as the

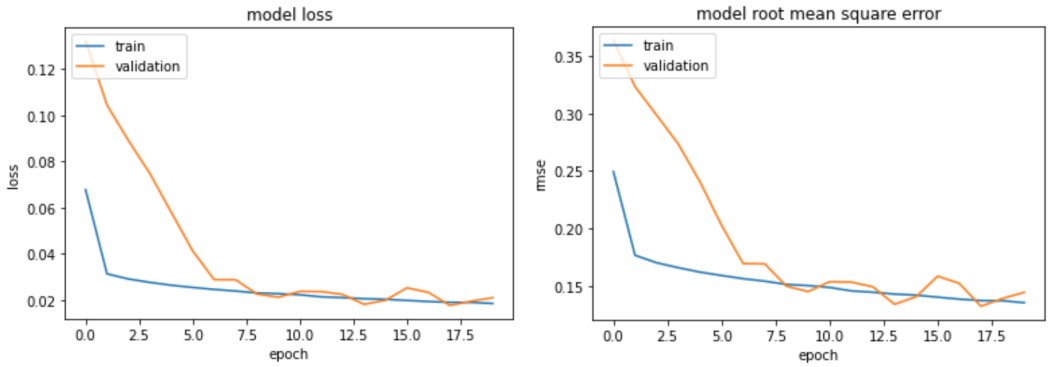
training one, but overall, it underwent a considerable reduction, since the loss in the last epoch had a value of 0.019.

For the error evolution, it was almost identical to the loss's. It started with an error of 0.2 and ended with an error of 0.1376. For the validation data, the final error's value was 0.1352.

For the evaluation on the test data, the prediction gave an error of 0.1369, also a very similar value to the validation data.

### 4.3 Third approach

The third model used the same augmented data as the second one but with a different structure, as presented in the section 2.3. The graphs produced by this model's training are the following:




(a) Loss evolution

(b) RMSE evolution

Figure 6: Loss and RMSE for the third model

The shown graph displays a slightly different performance than the previous models'. Here, the values for both the loss and the error start remarkably high, but they go through a drastic descent in the inmediate epochs. So from the epoch 10 point forward, the values don't get much better. This could lead to an overfitting because the model could learn in too much detail the training data.

About the scores, the validation of error gave a value of 0.1447, which is a bit higher than the previous ones. For the prediction of the test data, the model gave an error of 0.1464.

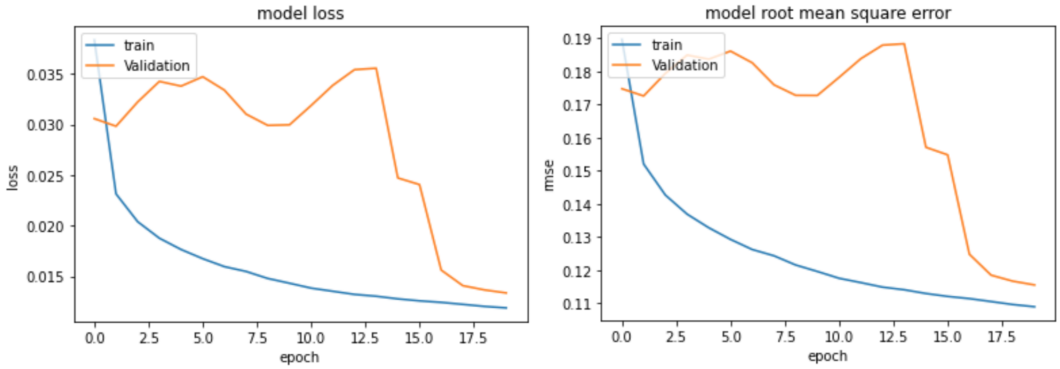
### 4.4 Fourth approach: ResNet

For this last approach, I chose to use a pre-trained ResNet model. The particular model I used was pretrained on the *imagenet* dataset. ImageNet is a project which aims to provide a large image database for research purposes and it contains more than 14 million images which belong to more than 20,000 classes.

The reason why I decided to try this approach is because using an artificial neural network that has already been designed and trained allows us to take advantage of what the model has already learned without having to develop it from scratch.

In order to take advantage of this pre-trained model, I have had to fine-tune it. For this, we have to take into account that the last layer would have previously been classifying another task different from this project's, so we need to remove it. After removing this, we want to add a new layer back that's purpose is this project's task. In addition to this, I have added a few layers before adding the output one.

After that, the only remaining task is to train it. The amount of time used for the training of this model was 7 hours. The graphs produced by this model's training are the following:




(a) Loss evolution

(b) RMSE evolution

Figure 7: Loss and RMSE for the fourth model

As it can be seen, this model's evolution differs noticeably from all the previous ones. The validation data's values diverge considerably from the training values, since the first layers were trained on a different dataset. However, when the training reaches the 13th epoch, the validation data's loss and error start to drop substantially, reaching a final score of 0.1155 for the root mean square error, the best score obtained among all the previous models.

As for the previous models, I also used the test data to predict the target values with this model. The error for this one was 0.1161, an also very consistent value.

## 5 Conclusions

To start with, I am going to state the differences among the results of the models. As commented before, the first two approaches had the aim of comparing the differences between using cropped images or not. As the results have been very similar, there are two options: the images were not sufficiently cropped, so there wasn't much useless data taken, or the cropping doesn't affect too much to the model's performance. In any case, at least the cropping zoomed the region of interest so that then, the images could be resized to a smaller scale than the previous approach. This way, the training time was reduced.

Among the models implemented from scratch, the best one was the second one, given that the first and second have almost identical error values, but the second one has faster training.

The ResNet though was the one which gave the best results, even though through the first epochs of the training the validation error was too high, it gave the best score when it reached the 20th epoch. So a thing to take into account is that the number of training epochs is very significant for the results, seeing that if I had stoped the training before the 13th epoch, even if the training time would have been highly reduced, the results would have been very poor.

So overall, it can be concluded that the Convolutional Neural Networks are one of the best options if we have to work with images, because all of the tried models gave good results. Also, the pre-trained models are a very valid choice for these types of tasks, considering the results I got from the pre-trained ResNet. Lastly, it is important to bear in mind that training Deep Neural Networks can be computationally expensive, so it is crucial to have patience while training models.

## References


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

[2] Ilya Sutskever Alex Krizhevsky and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *In Advances in neural information processing systems*, (pages 1097–1105), 2012.

[3] Jorge De La Calleja and Olac Fuentes. Machine learning and image analysis for morphological galaxy classification. *Monthly Notices of the Royal Astronomical Society*, (349(1):87–93), 2004.