Training model for SPIRAL – ELLIPTICAL classification

Objective

Our objective is to create a model that classifies galaxies most efficiently and consistently as Spirals, Ellipticals and Uncertain, to later apply this model to classify objects for the whole SPLUS catalog. In order to do that, we use Machine Learning technology - Keras and Tensorflow - two libraries in python that allow us to create machine-learning models known as Neural Networks.

Methods

In order to achieve our goals, we need to do several important steps:

- Inspect the data, to look if there are any problems with the dataset and classes.
- Process the input data.
- Process the dataset, balance the dataset and split train, test and validation sets to train the model.
- Define model architecture.
- Fit the model on the training set.
- Evaluate the model.

Processing input data and dataset

For this test we are going to use the galaxy_zoo2 table available at https://data.galaxyzoo.org/. This table has about 8900 objects of which about 8200 had corresponding FITS (Flexible Image Transport System) files. Besides that, the table also has Spiral and Elliptical classifications, with approximately 2600 Spirals and 1200 Ellipticals.

Although we want to create a model that classifies in three classes, we are going to create a model that makes binary decisions, that is, it is going to tell us if the galaxy is near the group 0 (spiral) or group 1 (elliptical) for example, and for the third class we believe that if the model makes a prediction that is not so close to any of the groups for example 0.4 or 0.6, we are going to an “area” where we define the third class, that is of undefined objects.

As we saw earlier, we have to balance the dataset and for that we reject about 1300 spirals, leaving 1300 spirals and 1200 ellipticals, thus making it more balanced in numbers.

Next step we have to generate the images from the FITS files. Note that we choose to not use directly the FITS for experimental purposes.

The images were generated using python astropy “make_lupton_rgb” function with different bands, using the fits. Here’s a snippet:

```
PATH_G = "/path/G_stamps"
PATH_R = "/path/R_stamps"
PATH_I = "/path/I_stamps"

image_fileG = get_pkg_data_filename(f'{PATH_G}/{name}_G.fits')
image_dataG = fits.getdata(image_fileG, ext=0)
image_fileR = get_pkg_data_filename(f'{PATH_R}/{name}_R.fits')
image_dataR = fits.getdata(image_fileR, ext=0)
image_fileI = get_pkg_data_filename(f'{PATH_I}/{name}_I.fits')
image_dataI = fits.getdata(image_fileI, ext=0)

image = make_lupton_rgb(image_dataI, image_dataR, image_dataG)
```

We put this through a loop and generated all IRG images, as an example. It is also possible to do other filter combinations for better visualization, or alternatively one may generate the images in other ways.

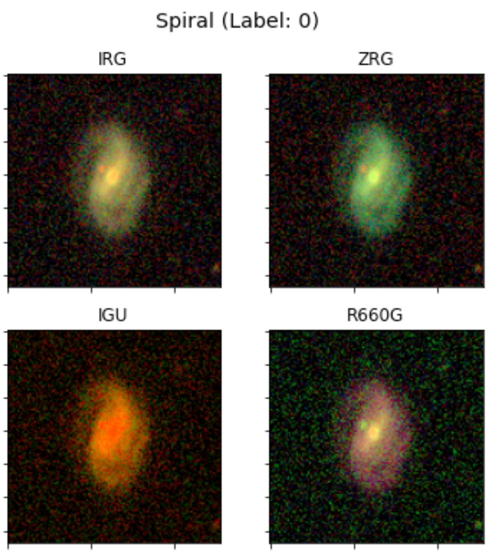


Fig. 1

For this project we tried the following combinations: IRG, IGU, ZRG and R660G.

Fig. 1 shows an example of a Spiral Galaxy in 4 different sets of bands (Img 1.00)

As can be seen there are certain combinations that show more clearly some features than others.

As for the training, validation and test set we divided 70%, 20% and 10% of the sample respectively.

We went through some difficulties throughout the process of selecting and processing the data. First the database had known issues. Some galaxies were very hard to classify by human eyes, so it was possible to have some misclassified galaxies and that was a problem, i.e., training with a wrong label leads to wrong results. And other thing was that the stretch (the linear stretch) of the images was often too high but luckily astropy "make_lupton_rgb" function had a parameter to adjust it.

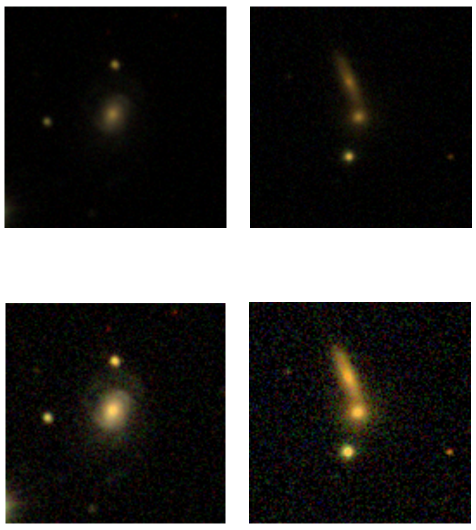


Fig. 2

In Fig. 2, it is possible to see the difference between the IRG set images that were generated with default parameters (on the top panels) and images that were considered better to see some features, using a different stretch (on the bottom panels).

Finally, we tested the influence of different sizes of cuts (image width and height), and it turned out that 128x128pixels images were the ones that worked best for most of the images. This image size was big enough to fit almost all objects, although leaving other objects that were around the main target, but still giving a clean view of the target.

Training

It is important to highlight that we tried different models with different inputs and the best ones were ones with the EfficientNetB7 customized by Prof. Clecio R. De Bom, and a simple top. EfficientNet is a network that relies on AutoML and other features to achieve superior performance without compromising resource efficiency.

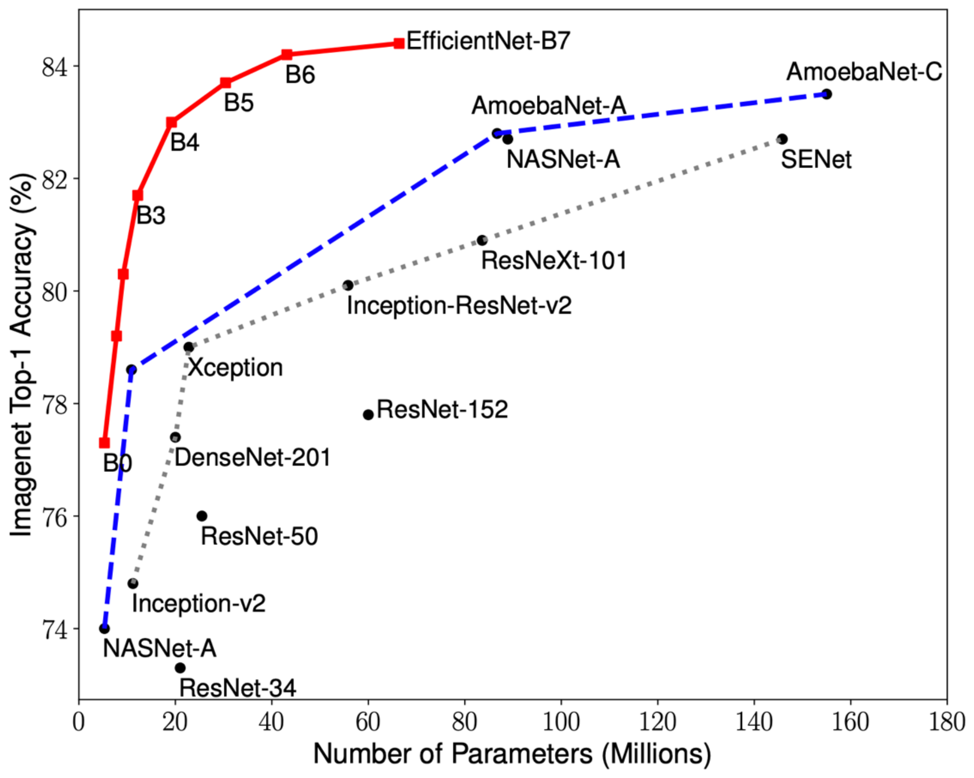


Fig. 3

In Fig. 3 we can see a comparison between the EfficientNets achieved accuracy on ImageNet (easily accessible image database) with a better efficiency comparing to other common models.

So, using this model as a pretrain and modifying the top we got the model below shown in Fig. 4:

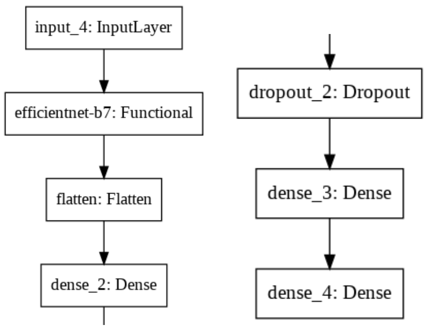


Fig. 4 - model, see text for details

Although it might not look like it is the case, it is a deep model because of the EfficientNet base and it fitted the samples very well as can be seen in the following sections.

Results

The results obtained with our best model got the following outcomes. First, we used a method to cross-validate the model called K-fold, that basically divides the model in N (in our case 5) parts and training the model for N (5) times with different train and test set organizations (Fig. 5).

This is an example of K-folding with K=5, that is divides the model in different organizations for different train runs and seeing how well it performs on each one.

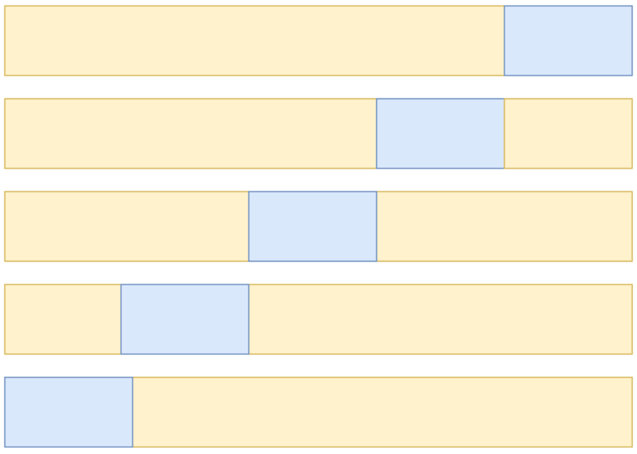
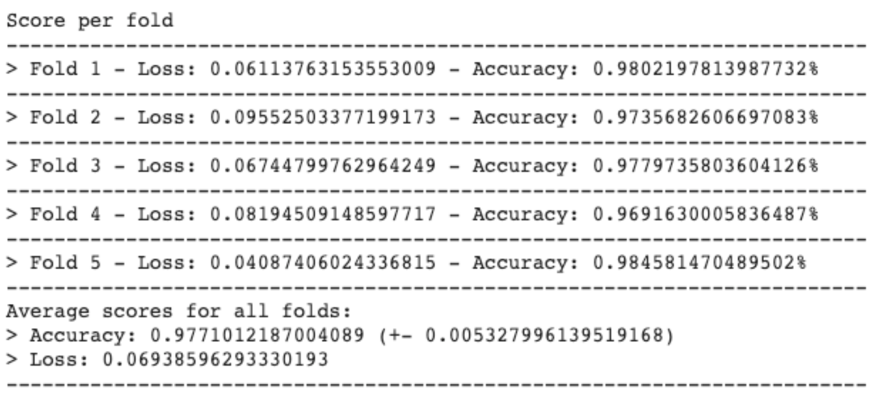


Fig. 5

For each split, the same model is trained, and later we can get the average scores across all folds to get a more realistic model evaluation than training just one time with just one organization of data.



```
Score per fold
------------------------------------------------------------------------
> Fold 1 - Loss: 0.06113763153553009 - Accuracy: 0.9802197813987732%
------------------------------------------------------------------------
> Fold 2 - Loss: 0.09552503377199173 - Accuracy: 0.9735682606697083%
------------------------------------------------------------------------
> Fold 3 - Loss: 0.06744799762964249 - Accuracy: 0.9779735803604126%
------------------------------------------------------------------------
> Fold 4 - Loss: 0.08194509148597717 - Accuracy: 0.9691630005836487%
------------------------------------------------------------------------
> Fold 5 - Loss: 0.04087406024336815 - Accuracy: 0.984581470489502%
------------------------------------------------------------------------
Average scores for all folds:
> Accuracy: 0.9771012187004089 (+- 0.005327996139519168)
> Loss: 0.06938596293330193
------------------------------------------------------------------------
```

Fig. 6

This is the evaluation across the five runs with different data splits (Fig. 6).

As can be seen in the Fig. 6, our model averaged 97.70% accuracy with 6.94% loss across all 5 folds. So, from this we took the model that performed the best judging validation score to make the tests.

First, we plotted the confusion matrix and the classification report with the test set of 250 images never seen by the model before, containing 110 Ellipticals, and 140 Spirals.

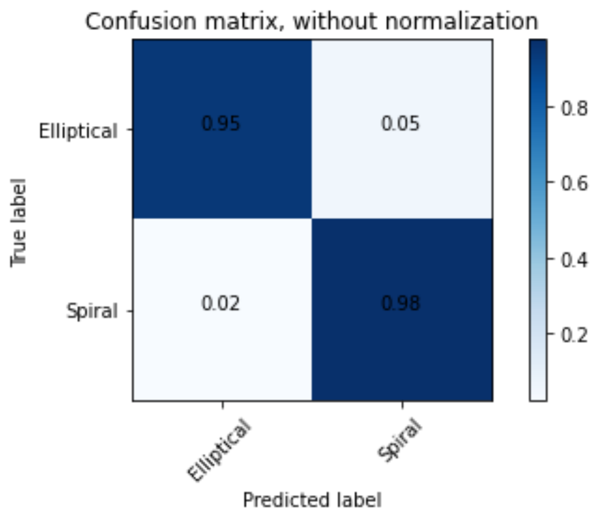
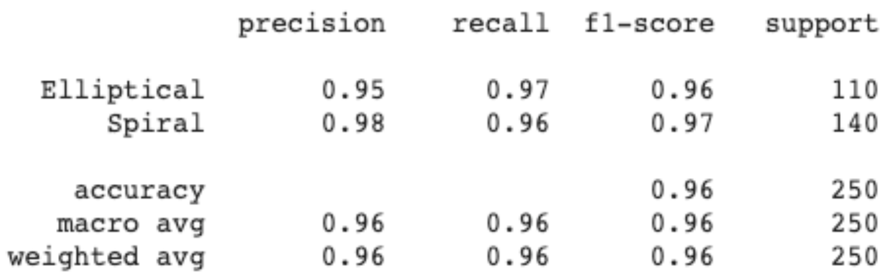


Fig. 7

As we can see in the Figs 7 and 8, the model performed well on both classes failing to predict around 3% of the objects.



```
              precision    recall  f1-score   support

  Elliptical       0.95      0.97      0.96       110
      Spiral       0.98      0.96      0.97       140

    accuracy                           0.96       250
   macro avg       0.96      0.96      0.96       250
weighted avg       0.96      0.96      0.96       250
```

Fig. 8

Moving on we plotted the ROC (receiver operating characteristic) curve as can be seen below in the Fig. 9. The ROC curve plots the true positive rate against the false positive rate, so the closer to 1 the area under the curve, the better the model is in theory.

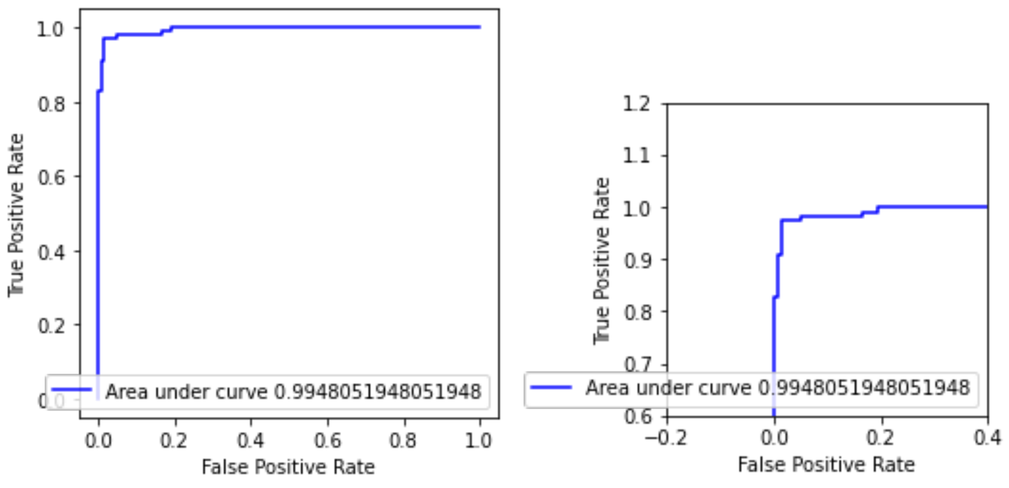


Fig. 9

We also plotted the Precision Recall (PR) curve since it is very similar to the ROC curve, however the PR shows some details and proves that we can use our theory to use a binary classifier to classify in three classes.

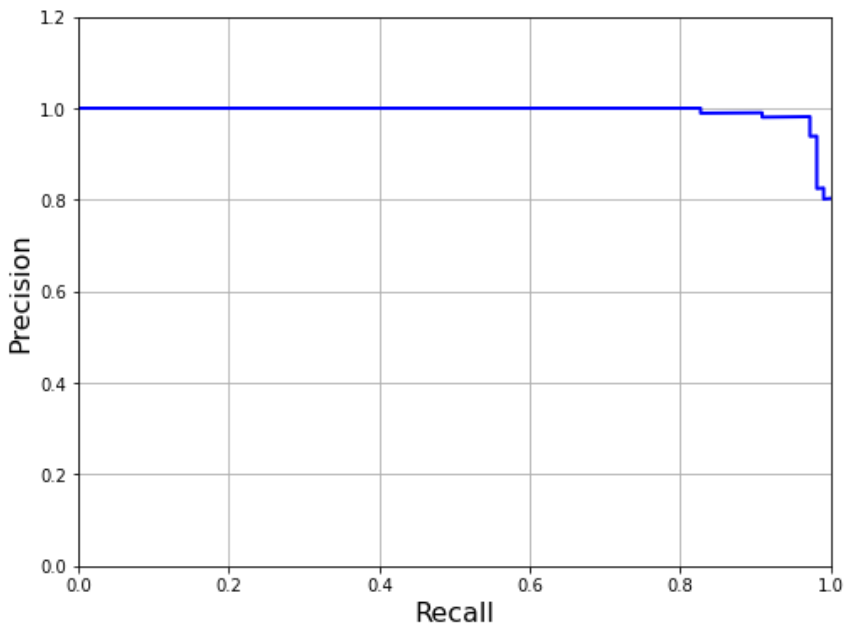


Fig. 10

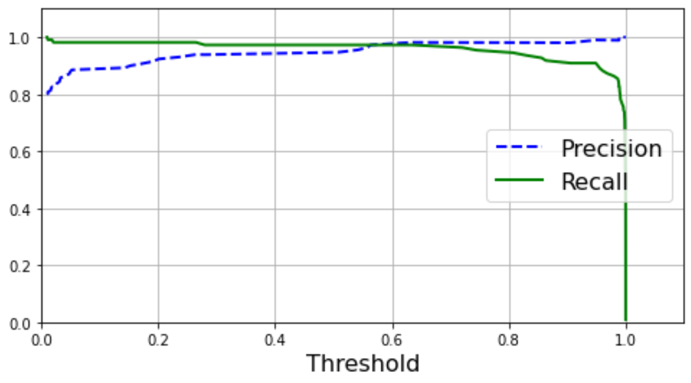


Fig. 11

As can be seen in Figs. 10 and 11, we have a very good precision recall trade-off. On Fig. 11 we can see that with about 0.85 threshold we have a precision of around 97-98% with a recall higher than 90%, that is a high precision with a high detection. Proving that the model has a certain assurance in its decisions, and maybe proving that we can use a threshold to define Undefined objects.

So, we decided to make some of the predictions that the model successfully made and saw that it was really predicting with assurance in most of the cases that the object was a spiral or elliptical.

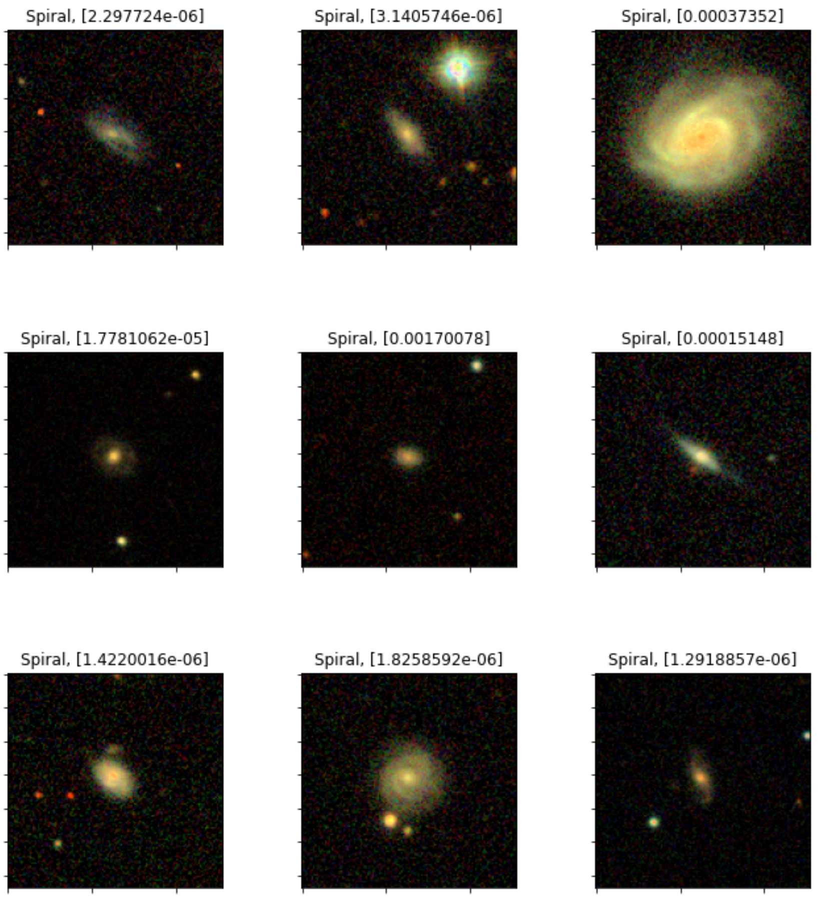


Fig.12

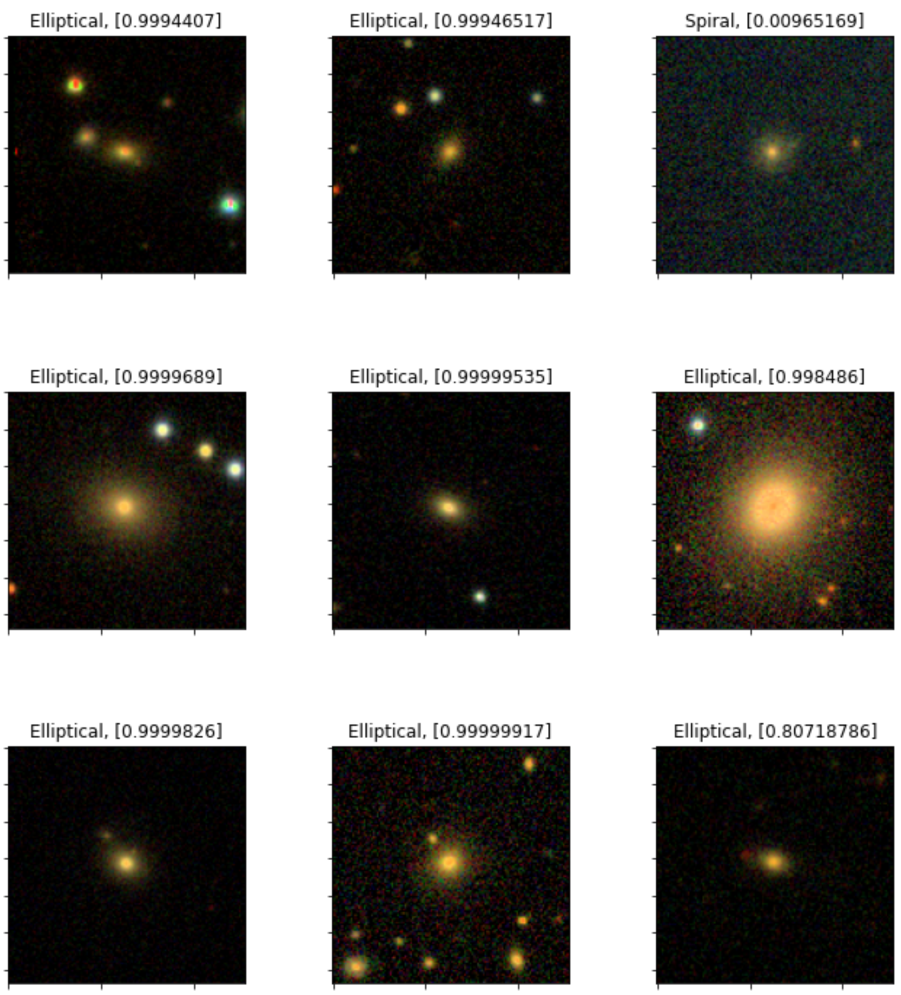


Fig. 13

As can be seen in the Figs. 12 and 13, the predictions that the model made on this spirals and elliptical objects never seen before were almost all very close to 1 or 0. So we decided to define objects between 0.25 and 0.75 as undefined and see the result. And testing with some objects defined as undefined by the zoo2 table to see how it performs and got the following results:

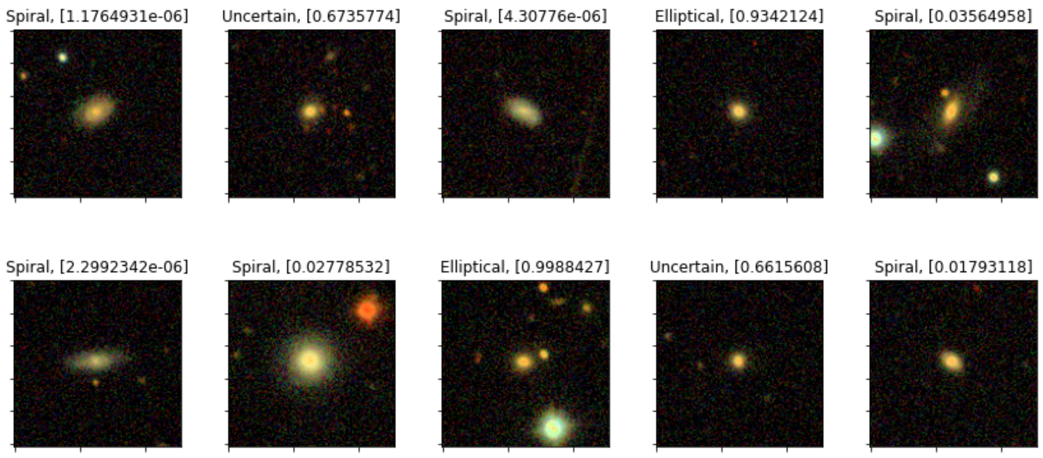


Fig. 14

As can be seen in Fig. 14 it performed very well, and some of the objects classified as undefined really are compacted galaxies or lenticulars, indicating that they are not elliptical or spirals.

In order to complete our tests, we used the model to predict the classification of the 1300 spirals that were left out for balancing purposes and the result was 97.5% accuracy as expected.