Project Report: Gamma-Ray Astrophisycs

Lorenzo Orsini
Gianluca Di Tuccio

Index

Abstract 3

1. Generate_sources.ipynb 4

2. TuningHyperparameters_test.ipynb 5

3. Example.ipynb 7

Abstract

Gamma-ray astronomy is the astronomical observation of gamma rays, the most energetic form of electromagnetic radiation. A gamma-ray transient of an astrophysical source is a variable behaviour, i.e. a significant change in the level of gamma-ray emission; in general, they are associated with catastrophic events involving compact objects, such as white dwarves, neutron stars, and black holes. To understand these phenomena, they must be observed during their evolution and communicated immediately to the international astrophysical community. For this reason, speed is crucial, and automated data analysis pipelines are developed to detect gamma-ray transients and generate science alerts (a significant detection of a transient source that requires a follow-up strategy) immediately.
In this project, we have developed an algorithm that classifies gamma-ray images generated via software for a given set of astrophysics parameters (see chapter 2). In particular, the algorithm was trained and tested with a dataset (360 images) composed of three types of images: images without gamma-ray sources (class 0), with only a source (class 1) and with two sources (class 2). For images with class 1, the algorithm outputs also the coordinates of the source, with a maximum error of 0.1°, with respect to the true coordinates, which are RA (right ascension) coordinate and DEC (declination) coordinate.
In the main folder there are three different notebook files:

- ***Generate_sources.ipynb***, which generates the gamma-ray images;
- ***TuningHyperparameters_test.ipynb***, which uses the generated dataset to tune the hyperparameters of the algorithm and evaluate its performance;
- ***Example.ipynb***, which performs an example of the algorithm for 3 images.

Also in the main folder, you can find the ***data folder*** (where image models are saved), ***models folder*** (where the .xml code for generating image models are saved), ***images folder*** (where the generated images are saved) and a "***coordinates.csv***", a table where the coordinates (if an image has a single source) and classes for each type of image are saved.

1. Generate_sources.ipynb

This file generates the gamma-ray images (fig.1.1) that will be used to train and test the dataset. First of all, we use the following astrophysics parameters (written in block 1.1 of the file) to generate the images of our dataset:

- ***energy*** (range from 0.1 TeV to 50 TeV);
- ***time interval*** (100s) of integration;
- ***irf*** (North_z20_0.5h), which contains templates that describe the spatial and spectral distribution of the background

Besides, we add other internal parameters, like the ***ROI*** (Region Of Interest), the pointing (where the telescope points) and the number of decimals for generating the coordinates of the sources. It is worth highlighting that these images report for each pixel the respective value of energy, which is a f*loat64* number, and not the grey level, which is a *uint8* number.

The program allows users to choose the size of the dataset, i.e. the number of images. We have used 360 images (120 for each type of image), which is the highest number of images that our available hardware can process. Due to the dimension of the dataset, we store this dataset on GitHub (*https://github.com/DitucSpa/ComputerVisionProject.git*), while in this project folder we store a dataset with only 36 images.

Thereafter, the manly purpose is to change the coordinates of the source (RA, DEC) into the ROI, thanks to the code model "***crab.xml***" (see block 1.3 of the Generate_sources.ipynb).

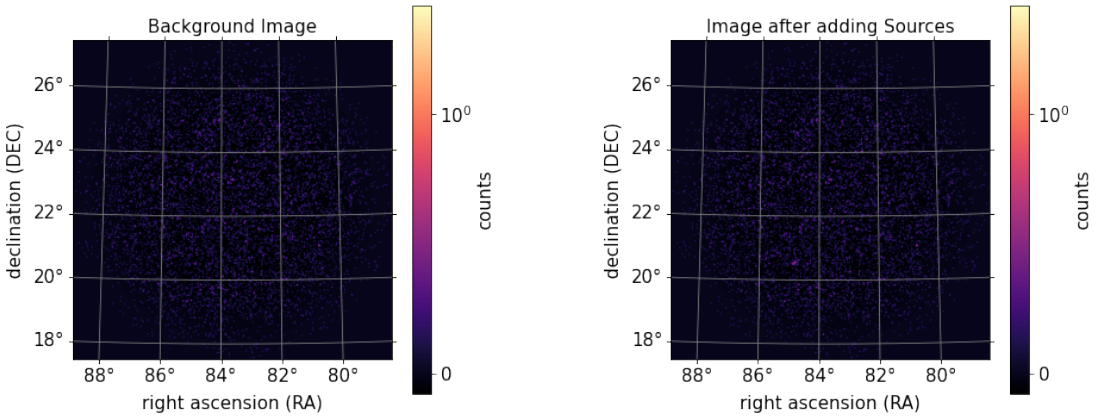

Figure 1.1: the left image represents the background with the astrophysics parameters (with a ROI of 5x5). The righ image represents the background image with the source (in this case the coordinates of the source are *RA = 84.7563* and *DEC = 20.5769*).

2. TuningHyperparameters_test.ipynb

This file uses the generated dataset to tune the hyperparameters of the algorithm and evaluate its performance. The dataset used in this project is composed of 360 images, with 120 images for each class: no source (class 0), one source (class 1), more than one source (class 2). We also used a table to store the coordinates of the one-source images, while in the other two cases it reported the value 'NaN'. Then we split the dataset into training and test set, whose dimensions are respectively 0.7 and 0.3, by keeping the same proportion of the classes for both sets. The tuning process was performed by considering a grid of values for each hyperparameter, and then running the algorithm for each possible pair. After that, the algorithm starts: firstly, each image is convolved with a Gaussian Kernel, which is defined by ***sigma = 4***, in order to perform smoothing. This filter is applied to the original image, whose pixels report the energy values (a float64 number). For this reason, we used the Gaussian Kernel of the 'astropy.convolution' library, which can deal with the float64 images. Then, the float64 value of each pixel is converted to a uint8 type through a linear mapping, defined by the equation:

$$u = \frac{(u_{max} - u_{min})}{(f_{max} - f_{min})} \cdot (f - f_{min}) + u_{min}$$

where f_max and f_min are respectively the max and min value of energy, while u_max = 255 and u_min = 0. So the highest values of energy correspond to pixels with the brightest values, while the lowest ones correspond to the darkest. In this way, we have obtained an image whose pixels values are grey levels between 0 and 255. This step was performed in order to obtain unit8 NumPy that is compatible with the traditional OpenCV functions; this mapping follows the filtering step to obtain unsmoothed uint8 values. After that the algorithm performs a thresholding step, in order to detect the presence of a source, by using as threshold the median of the grey level histogram multiplied by a factor, which is one of the two hyperparameters of the model. We use the median due to the peculiarities showed by the grey-levels histograms of the different class of images (fig. 2.1): in fact the pixels of a gamma-ray source exhibit higher energy values than any others in the image, which correspond to the higher levels of grey in the histogram. For this reason, we use the median multiplied by a factor, called **median_factor**, as a way to detect the set of pixels with the highest values. The tuning of this factor has been performed in a range between 2.2 and 2.8.

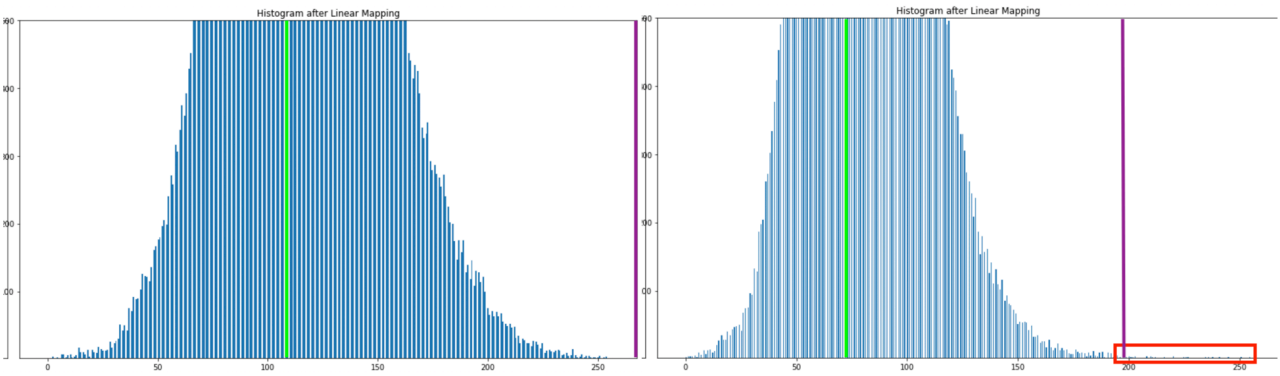

Figure 2.1: A grey levels histogram for an image without sources (on the left) and with a source (on the right). As we can see, the presence of a source implies that there will be few pixels with the highest grey levels (highlighted by the red box) than the images without sources. Besides the green line represents the median of the image, while the violet line the median multiplied by the factor.

So this step set to 255 all the pixels that satisfy the threshold (foreground pixels) and the others to 0 (background pixels); otherwise all the grey levels in the image are set to 0, to highlight the absence of a source.
In the latter case, the algorithm assigns to the image class 0, which means that no sources are detected. In the former case, the thresholding step is followed by the labelling and blob analyzing step. The labelling is performed by assigning to all connected regions the same integer value. In particular, the algorithm keeps only the connected region with a number of pixels that are at least equal to **n_pixels**, which is the other hyperparameter. This one has been tuned in a range between 3 and 9. If the unique labels are more than 2 is assigned class 2 to the image, because the algorithm has found two sources. In the last case, we have only labels 0 and 1, so the algorithm finds the contours of the blob and then draws its enclosing circle, taking as coordinates the centre of this circle. At the end of the tuning step it is printed the classification report for each pair of hyperparametyers, selecting the pair with the highest macro F1 score, which correspond to:

	precision	recall	f1-score	support
Class 0 [No Sources]	0.98	1.00	0.99	84
Class 1 [One Sources]	0.91	0.96	0.94	84
Class 2 [Multiple Sources]	0.99	0.90	0.94	83
accuracy			0.96	251
macro avg	0.96	0.96	0.96	251
weighted avg	0.96	0.96	0.96	251

```
The percentage of images in which the coordinates don't exceed the error max is: 100.0%
Median Factor:  2.9, N. Pixel:  10
```

Figura 2.2: classification report for the best pair of hyperparameters.

So the best hyperparameters found are: Median_factor = 2.9 and n_pixel = 10.
In addition, foe each pair of hyperparameters is reported the percentage of images with a source in which the coordinates found exceed the maximum error. The number of these images is printed after the classification report. After the tuning step the algorithm was run on the test set, using the two optimal hyperparameters found previously. At the end, the classification report on the test set was the following (figure 2.3):

	precision	recall	f1-score	support
Class 0 [No Sources]	0.97	1.00	0.99	36
Class 1 [One Source]	0.87	0.94	0.91	36
Class 2 [Multiple Sources]	0.97	0.86	0.91	37
accuracy			0.94	109
macro avg	0.94	0.94	0.94	109
weighted avg	0.94	0.94	0.94	109

```
The percentage of images in which the coordinates don't exceed the error max is: 100.0%
```

Figure 2.3: classification report on the test set using the best pair of hyperparameters

3. Example.ipynb

In this notebook file we runned the algorithm for the 3 different types of images, by using the optimal hyperparameters found previously on the dataset of 360 images. For each example are printed:

- the original image;
- the image after the application of a Gaussian filter;
- the reuslting grey-levels histogram;
- the grey level histogram after the linear mapping from float64 to unit8;
- the resulting image after the previous mapping;
- the image after the thresholding step.

Here we can see an example for an image with a source:

```
MODEL 58
error:  0.01104 degrees

LABEL 0              true coordinates                predicted coordinates
degrees:          81.32756,    21.84801             81.33860,    21.84180
```