Data Science

Proyect: Astrodetector

Alberto Guirado Fernández, Student ID: 82122300060

Ritsumeikan University, 2024

Introduction

Deep learning is a powerful tool for the study of space. In astrophysics, various artificial intelligence methods are used to detect and study the innermost elements of the cosmos. These studies allow us to understand how our reality works and how objects interact in the macroscopic world.

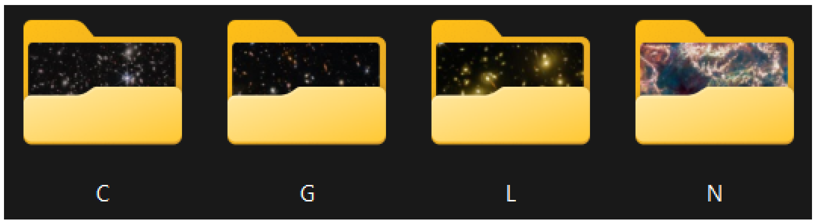
The problem we are going to tackle has focused on collecting a set of images obtained through various telescopes around the world and beyond, such as Hubble and James Webb. These images contain characteristic elements of the cosmos, such as galaxies, nebulae, gravitational lenses and clusters.

Our programme will recognise and classify each of the images into these four groups by training a deep learning model. The images will be divided into two groups: the training group and the test group. Different divisions will be made to assign different percentages to each group and see how the algorithm performs.

Dataset

The dataset is composed of images of 4 types: clusters, galaxies, gravitational lenses and nebulae. These images have been obtained by different ground-based and space-based telescopes and then uploaded to different websites, from where they have been compiled for the construction of a more complex dataset.

In the algorithm, each image is reduced to a certain size for evaluation in vector form.



What is a nebula

Nebula or nebulae are regions of the interstellar medium made up of **gases**, mainly hydrogen and helium, as well as chemical elements in the form of cosmic dust. They have a notable cosmological importance because many of them are the places where stars are born due to phenomena of condensation and aggregation of matter; On other occasions it is the remains of already extinct or dying stars.

They are characterised by the composition of the gas, the colour they emit in the different ranges of the visible spectrum and their irregular shape.

What is a galaxy

A galaxy is a collection of stars, gas clouds, planets, cosmic dust, dark matter and energy gravitationally bound together in a more or less defined structure.

They are characterised by a large number of stars at the same point. There are several types depending on their shape, which can be spiral with different "galactic arms". This shape is given by the rotation around a hypermassive body such as a black hole.

What is a gravitational lense

One of the rarest and most interesting objects in the sky. They are a phenomenon predicted by Einstein's theory of general relativity, in which the presence of mass and energy in space bends the time and space around them. The study of these phenomena may provide us with answers about **dark matter**.

They are characterised by having a circular or ring shape which organizes a central element such as a star or a black hole.

What is a cluster

A globular cluster is a spherical group of stars that generally orbits a galactic nucleus as if it were a satellite.

They are characterised by being a large set of bright points that bring together the same space in a very close manner.

Types of data

En nuestro dataset como hemos visto, tenemos 4 tipos de datos los cuales serán nuestro label objetivo en el entrenamiento del modelo.

ID	IMAGE PATH	TYPE
1	DATASET/L/L-1	LENSE
194	DATASET/C/C-7	CLUSTER
385	DATASET/N/N-1	NEBULA
786	DATASET/G/G-123	GALAXY

As mentioned above, the dataset has been built through different web images that have been obtained through web scraping.

An analysis of the source code of the HTML page was carried out to obtain only the part where the image was found and a script was created to obtain it.

The web scraping script iterates through the different images to automate the download of each of these images.

```python
url <- "https://research.ast.cam.ac.uk/lensedquasars/"

response <- requests.get(url)

if response.status_code == 200:

  soup <- BeautifulSoup(response.text)

  img_tags <- soup.find('img')

  counter = #number

  for img_tag in img_tags:

    img_url <- urljoin(url)
    img_data <- download(img_url)

    img_filename <- "IMG{counter}.jpg"

    counter += 1
```

For its use in the dataset and to have homogeneity, a script has been carried out to rename each of the files in their respective folders.

```
Python

def rename(mainFolder):

  for folder in mainFolder:

     files <- [f for f in folder

      for i, file in files:

         name, extension = file

         newName <- "folder-i.jpg"

         path <- os.path.join(folder, file)

         newPath= os.path.join(folder, newName)

         os.rename(path,newPath)
```

Files

GuiradoFernandezAlberto_dsfinal → Algorithm that contains the functions responsible for executing the model. Split data into training set and test set.

Dataset → Folder that contains all the data classified into subfolders.

Dataset/{C,G,L,N} → subfolders that contains all the specific data

Dataset/{C,G,L,N}/{G_x.jpg} → Data in form of images jpg.

Parameters

- ***random*** → Functions related to random number generation
- **numpy as np** → Numerical library for **mathematical operations**
- **os** # Functions related to the **operating system**
- **pandas as pd** → Library for **data manipulation** and analysis of tabular data
- **seaborn as sns** → Library for statistical data **visualisation**
- **matplotlib.pyplot as plt** → Library for **plotting** charts
- **display, Image as IPImage** → Functions for **displaying images** in the IPython environment
- **train_test_split** → Function for **splitting** data into training and testing sets
- **LabelEncoder, OneHotEncoder** → Functions for **encoding** categorical variables
- **RandomForestClassifier** → Random Forest **Classifier**
- **classification_report** → Function for reporting **classification metrics**
- **Image** → Library for **image manipulation**
- **resize** → Function for **resizing** images
- **train_test_split** → Split dataset into **training** and **testing** sets
- **Layers** → Layers for building **neural networks**
- **Sequential** → Linear stack of **layers**

Methodology

For the **identification** and **classification** of the elements of our dataset, two models have been built: one based on neural networks thanks to the **Keras** library and another using the **RandomForestClassifier** library.

The Keras library allows the creation of a neural network model in which we declare the layers one by one with specific characteristics.

Keras

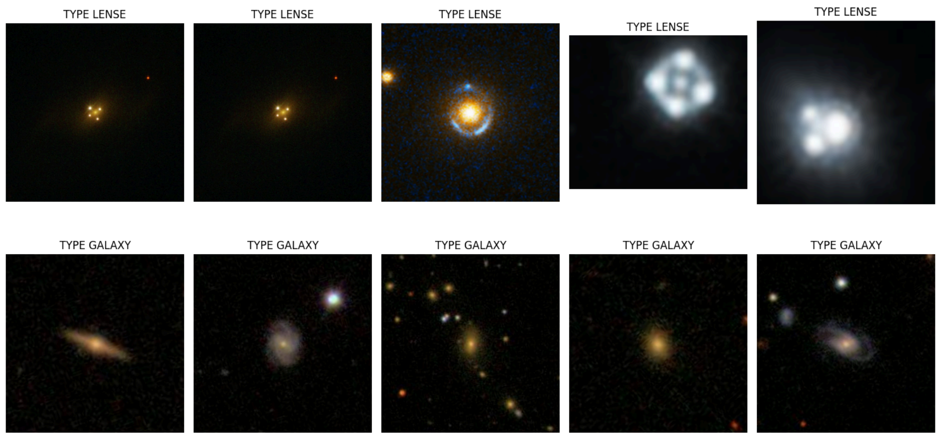
Firstly, our dataset is loaded for prior visualization and processing before converting it into input for the neural network. It is checked that it does not have any empty datapoint.

Our dataset is composed of +7000 images and four types. In our data we find a disproportion of quantities since one of the types composes 90% of the dataset. To balance our dataset we will limit the number of this type in order not to overload the model with only one type.

We set a limit on the number of images we are going to deal with for that type and build a new dataset, dumping all the data with this new limitation..

```
Python

galaxy_limit = 300

count_galaxy = df['Type' == 'GALAXY']

if count_galaxy > galaxy_limit:

    df_galaxy <- df['TIPO'] == 'GALAXY']

    df_no_galaxy <- df[df['TIPO'] != 'GALAXY']

    df_balanced <- pd.concat([df_no_galaxy, df_galaxy])

  df2 = df_balanced
```

With our new dataset, we are going to transform and visualise some of the data along with its label. We create a template where we place the resized images.



Training set and Test set

In order for the model to be able to process the data: images and labels, it is necessary to transform them. In the case of labels, they are transformed by means of the LabelEncoder function into integers for their handling.

The dataset is separated into training and test sets by a specific percentage. In our case 70-80%.

```python
train_df, test_df = train_test_split(df2, test_size=0.3, random_state=SEED)
```

The dataset is divided into images and labels for model training. This is done by the function data_preprocessing(...).

The data processing function receives the training set with images and labels and separates it into two sets: X and y, being X the information of the images and y the information of the labels.

Within this function we define the size of the images. This is important because if they are too small the model will not be able to detect the patterns that characterise each type. On the other hand, if they are too large, the execution time may be too long.

For better convergence and to treat the data uniformly, the image is normalised by dividing by 255 to scale the values of each pixel.

```
Python

def data_preprocessing(df):
  df["TIPO"] <- label_encoder.fit_transform(df["TIPO"])
  labels  <- to_categorical(df["TIPO"])
  images  <- []
  for index, row in df:
    img <- keras.load_img(img_path, color_mode='grayscale', target_size=(100, 100))
    img_array <- tf.keras.preprocessing.image.img_to_array(img)
    img_array /= 255.0 # Normalize
    images  <- img_array
  tensor(images )
  return images , labels
```

Model

When the data is suitable for processing, the neural network model is built. In this case it is a Sequential model with different layers:

1. The first layer, a **convolution layer**, has 16 filters. Each filter is a three-dimensional matrix that runs through the image to perform the convolution operation. Has Kernel of 5x5 which indicates the size of the matrix used in convolution. A ReLU activation and a stride of 2, which is the number of pixels that the kernel moves horizontally in the image. The padding="same": padding is added to the input to maintain the same height and width as in the input and its shape is (100,100,1), which is the size of the images the model will work with.
    a. The ***ReLU*** (Rectified Linear Unit) activation function is a non-linear function defined as f(x) = max(0,x). This function introduces non-linearities into the network, which allows for the learning of more complex patterns and representations of the data.
    b. Its output is 0 for all negative values and is equal to the input for non-negative values. That is, it produces non-zero output with positive input and zero output if the input is negative.

2. The **MaxPool2D layer** consists of a pooling layer which has a window size of 2x2 on which the spatial dimensions of the representation of the previous layer will be reduced. The default strides and padding are the same as the previous layer.
3. The third and fourth Conv2D are convolution layers with 32 filters, a kernel size of 3x3, ReLU activation, only one strides and the padding in valid
4. The **Flatten layer** is responsible for "flattening" the input, i.e. “unidimensionalising” the multidimensional input, which is commonly used in the transition from the convolutional layer to the fully connected layer. Coupling does not affect the batch size.
5. Two **Dropout layers** for regularization
6. A **Dense layer** with 128 filters or neurons using relu activation
7. The last **output Dense layer** has as many units as we have classes in the classification problem (label_encoder.classes), in our case four: (Galaxy, Lense, Nebula, Cluster). The softmax function is used as it is commonly used for multi-class classification problems.
    a. The **softmax** function is in charge of transforming vectors of real numbers into a vector of probabilities, where the sum of these is 1.
    b. In our problem, the output layer uses this function to produce the probabilities associated with its class. The class with the highest probability is considered the final prediction of the network.

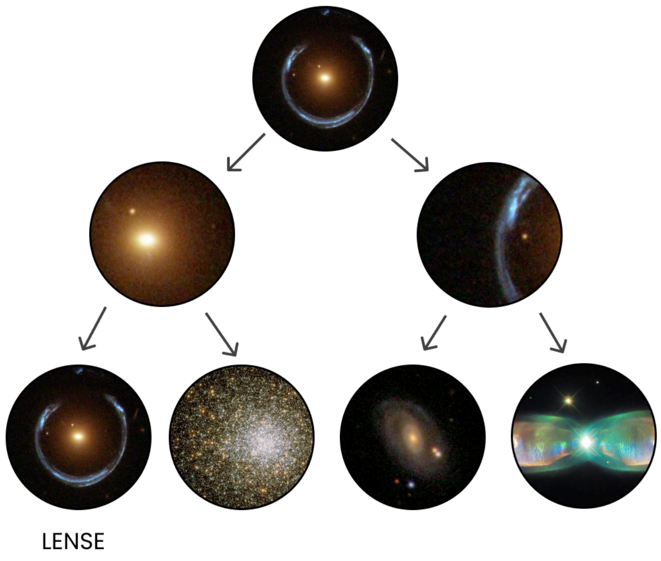
Once we have all the layers of the model, we need to compile it for training. For this purpose, the optimiser 'Adam' is used. The optimiser is of great importance because it is responsible for generating better weights. Its operation is based on the calculation of the gradient of the cost function for each weight (parameter) of the network.

In essence, the goal of training neural networks is to minimise the cost function by finding suitable weights for the edges of the network, ensuring good generalisation.

RandomForestClassificator

The **RandomForestClassificator** model is a model which uses tree structures for classification. Multiple decision trees make different decisions on random subsets of data and features. Each tree offers an answer and a prediction. The final answer is taken from the most popular prediction result.

For the model with RFC we also take the images of the dataset and the transformation of the images is carried out individually. In our case they are resized to 500x500 to show them with better quality and detail but later they are reduced for the model to 100x100.



Images are flattened by calling the flatten() function into a one-dimensional format. This is necessary when working with algorithms or models that expect inputs of this type instead of images directly. For example, if we have a 100x100 image, it will become 100x100x784.

Next, the image matrix X and the label vector y are created to separate the training data and test data. We have the same partition as in the Keras model, 70% for training and 30% for testset

The image data must be resized again to apply it to the model.

Experiments

To see the difference between these two models, they are run in different configurations:

Keras

With the data transformed and adapted for the model, we performed the compilation. As mentioned above, we use the optimiser Adam

```python
model.compile(optimizer="Adam", loss="categorical_crossentropy",
                metrics=['accuracy'])
history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
          epochs=80)
```

For a better analysis, we check the training time.

Forest

When we have the data prepared and transformed, we can train the model. Using the RandomForestClassifier call we can assign different parameters:

- ***Criterion*** → Used to specify the function to be used to measure the amount of splitting. The variable entropy is used to use the property of the same name to measure the purity of the divisions of the decision trees.
- ***Max_features*** → Used to specify the features to be taken into account for the best splitting.
- **N_estimators** → Number of trees in the forest.
- **Verbose**: Information that the algorithm provides during training.

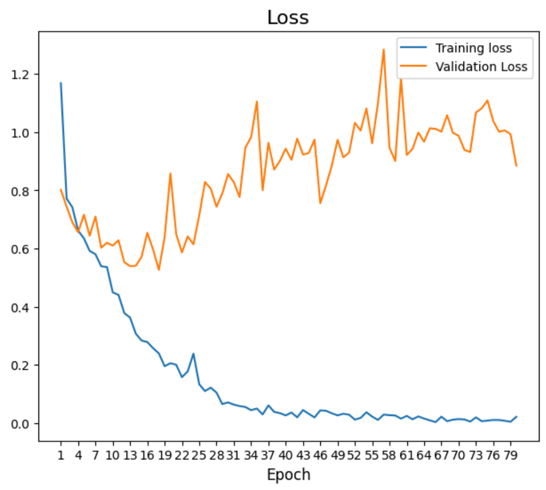
Results

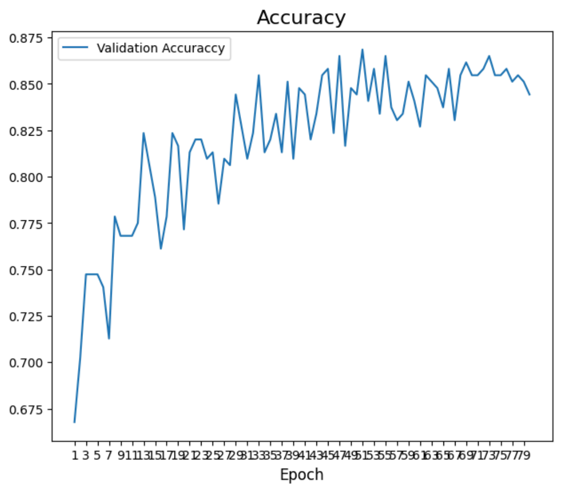
Keras

The results obtained are as follows:

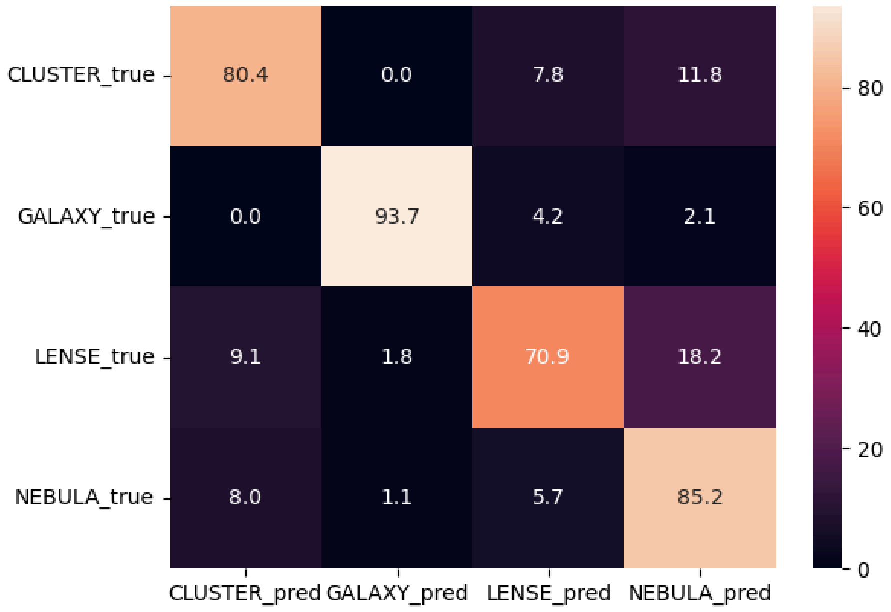
We can see that in the iterations the value of validaton_loss as well as that of accuracy are increasing. Specifically, in the accuracy we can see how it reaches an asymptote in which it is maintained. This is due to the fact that the model is converging in the learning process. Execution time: 17 seconds

KERAS				
	Precision	Recall	F1-Score	Support
CLUSTER	0,77	0,8	0.79	51
GALAXY	0.98	0.94	0.96	95
LENSE	0.75	0.71	0.73	55
NEBULA	0.81	0.85	0.83	88
Accuracy			**0.84**	289
Macro Avg	0.83	0.83	0.83	289
Weighted Avg	0.83	0.83	0.83	289





In the heatmap we can see that the best detected class is the GALAXY class. This is because the galaxy dataset is the most homogeneous of the 4 datasets and has the most similar elements to each other.

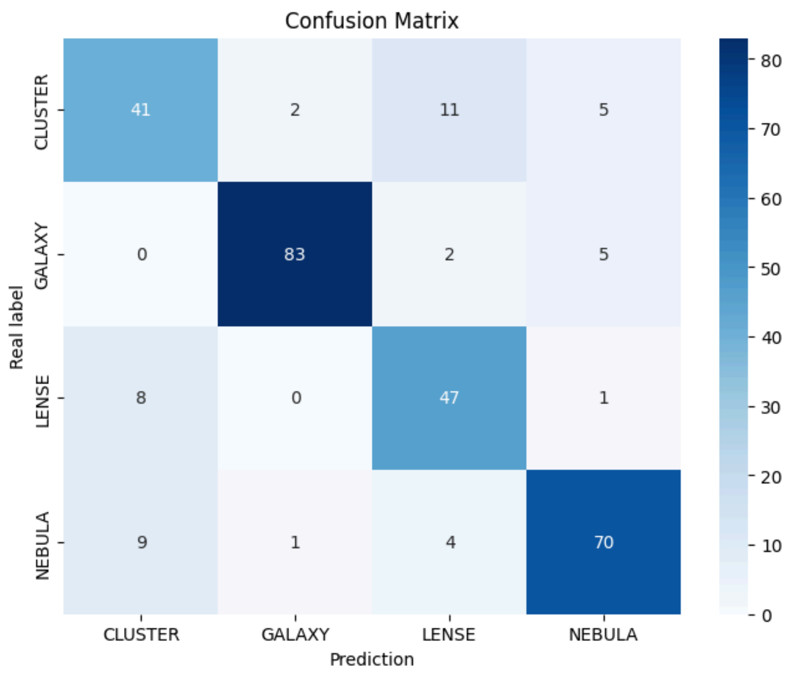


Forest

The results of the second model are similar to the Keras model. We see that the best predicted class is galaxies.

Running time: 4 minutes and 30 seconds.

RandomForestClassifier				
	Precision	Recall	F1-Score	Support
CLUSTER	0,74	0,73	0.74	59
GALAXY	0.94	0.94	0.94	90
LENSE	0.68	0.77	0.73	56
NEBULA	0.92	0.85	0.83	84
Accuracy			**0.84**	289
Macro Avg	0.83	0.82	0.82	289
Weighted Avg	0.83	0.83	0.84	289



To check the validity of the model we have to load it as "*astrodetector.model*". We must present an image that is properly configured, that is, like the images that I took for learning, so it must be resized.

```
Python
def prepare(file):
  size <- 100
  img_array <- cv2.read(file, cv2.IMREAD_GRAYSCALE)
  img <-  cv2.resize(img_array, (size))
  return img
```

After this, we load the image and deliver it to the model

```
Python

pred = model.predict([prepare("nebula.jpg")])

predicted_class = tf.argmax(pred, axis=1).numpy()[0]

predicted_label = CATEGORIES[predicted_class]

print(pred)
```

Predictions

PREDICTION	OUTPUT
pred = model.predict([prepare("nebula.jpg")])	[[0. 0. 0. 1.]] Predicted class: 3 TYPE: NEBULA
pred = model.predict([prepare("nebula2.jpg")])	[[0. 0. 0. 1.]] Predicted class: 3 TYPE: NEBULA
pred = model.predict([prepare("cluster.jpg")])	[[1. 0. 0. 0.]] Predicted class: 0 TYPE: CLUSTER
pred = model.predict([prepare("lense.jpg")])	[[0. 0. 1. 0.]] Predicted class: 2 TYPE: LENSE
pred = model.predict([prepare("galaxy.jpg")])	[[0. 0. 1. 0.]] Predicted class: 2 TYPE: LENSE

As we see, some of the predictions have flaws since it confuses some of the input images due to the similarities of the elements.

Conclusions

As we can see, the results obtained with the two models are similar. This may be due to the similarity in the way the models are trained or to the way the dataset is constructed.

The main differences between these two models is that with Keras we are creating a CNN (convolutional neural network) model which uses convolutional layers, pooling layers, regularization layers... On the other hand, the RandomForest model uses a random forest algorithm to perform the classification. As already mentioned, this model combines multiple trees trained on different subsets of data to give a prediction output.

We can say that both models have obtained a good overall precision, being 84%. The RF model has significantly better predicted the GALAXY and NEBULA classes with a higher accuracy than the other two classes. It can be considered a more interpretable model. The Keras model also shows good performance for the GALAXY class. This neural network model can learn more complex and non-linear patterns in images, depending of course on their size.

We can say that the most efficient model is Keras as it has better precision in each model as reflected in the heatmap and because it has a shorter training time of 17 seconds compared to 4 and a half minutes for the Random Forest Classifier.