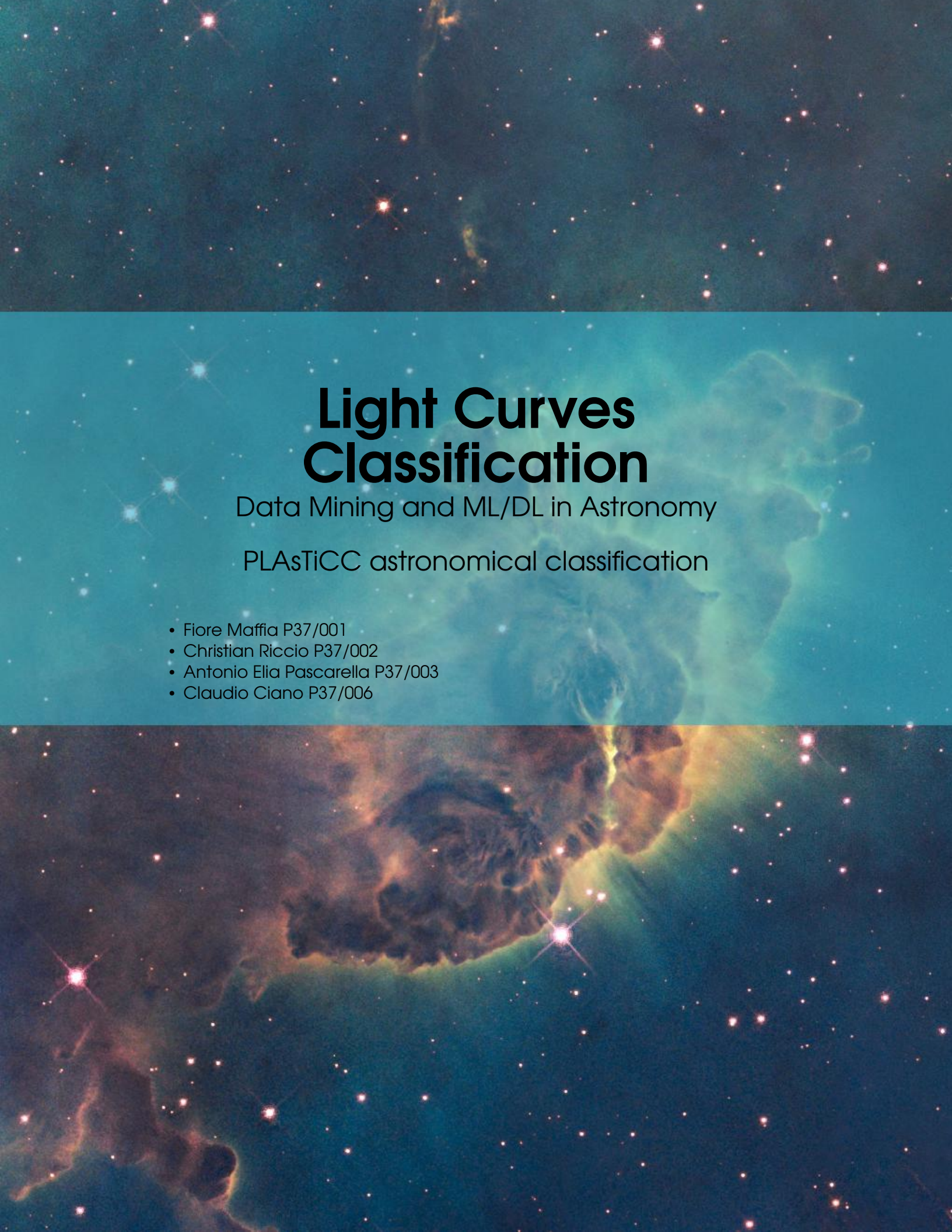


Light Curves Classification

Data Mining and ML/DL in Astronomy

PLAsTiCC astronomical classification

- Fiore Maffia P37/001
- Christian Riccio P37/002
- Antonio Elia Pascarella P37/003
- Claudio Ciano P37/006

Contents

1. Introduction

1.1 Motivation

How well can we classify objects in the sky that vary in brightness from simulated LSST time-series data, with all its challenges of non-representativity? Our work's aim was to develop a system for the light curves classification within the context of the *PLAsTiCC astronomical challenge*. After a strong work of data pre-processing we proposed different deep learning architectures for the required classification tasks.

> This project is based on a challenge proposed on Kaggle.com in 2018, with the goal of helping some of the world's leading astronomers grasping the deepest properties of the universe. The data that we used in this project have been retrieved from this challenge.

> Historically, the human eye has been the arbiter for the classification of astronomical sources in the night sky for hundreds of years. But a new facility – the Large Synoptic Survey Telescope (LSST) – is about to revolutionize the field, discovering 10 to 100 times more astronomical sources that vary in the night sky than we've ever known. Some of these sources will be completely unprecedented!
>
> The Photometric LSST Astronomical Time-Series Classification Challenge (PLAsTiCC) asked to scientists across the world to help classifying the data from this new survey.The goal is to classify astronomical sources that vary with time into different classes, scaling from a small training set to a very large test set of the type the LSST will discover.

The Large Synoptic Survey Telescope (LSST), has achieved first light in 2019 and will commence its 10-year main survey in 2022. LSST will revolutionize our understanding of the changing sky, discovering and measuring millions of time-varying objects.

These simulated time series, or 'light curves', are measurements of an object's brightness as a function of time - by measuring the photon flux in six different astronomical filters (i.e. passbands). These passbands include ultra-violet, optical and infrared regions of the light spectrum.

The simulated objects have been divided into 14 different classes and each class is identified by a number. We have to analyze each set of light curves and determine a number that represents the probability that each object belongs to each of these classes.

1.2 Context

Figure 1.1: Position of the LSST on the Chile coast

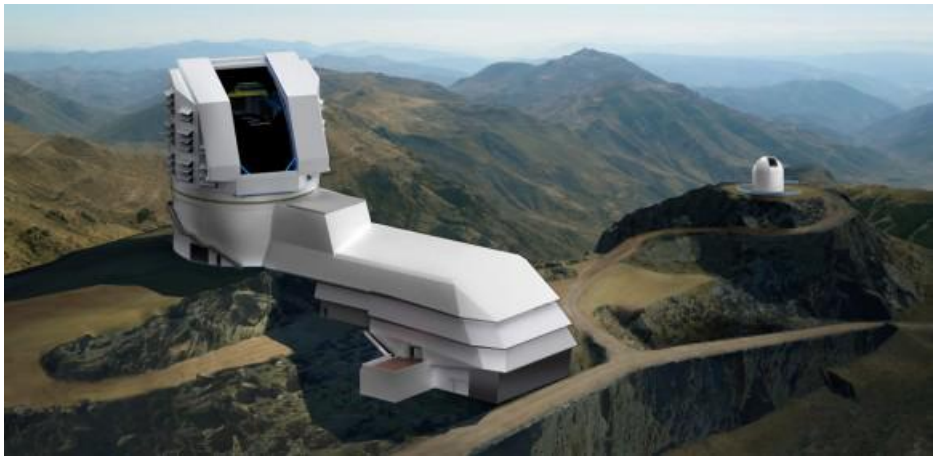


Figure 1.2: LSST picture

The Large Synoptic Survey [Figure 1.2] Telescope will use an 8 meter telescope, with a 3 billion pixel camera to picture the entire Southern sky roughly every few nights and over a ten-year duration. The telescope is located on the *El Penon* peak of Cerro Pachón, a 2,682-meter-high mountain in Coquimbo Region,in northern Chile [Figure 1.1], alongside the existing Gemini South and Southern Astrophysical Research Telescopes.

1.3 Deeper in detail

These light curves are the result of difference imaging: two images are taken of the same region on different nights, and the images are subtracted from each other. This procedure catches object that stay at the same position but vary in brightness. Depending on when the object first exploded or brightened, the flux may be increasing or decreasing with time. The specific manner in which the flux changes is a good indicator of the underlying type of the object. We use these light curves to classify the variable sources from LSST. The data are classified using a set of training data (light curves) for which the true classifications (i.e. labels) are given, also we create a validation test used to monitor the performance of the classifier real time and then, finally, there is the performances evaluation of the models on the test set.

1.4 Astronomy Background

As we might imagine, sky is filled with sources of light that vary in brightness on timescales from seconds and minutes to months and years. These events are called:

- **transients**
- **variables**

For example, the cataclysmic event that occurs when a star explodes generates a bright 'supernova' signal that fades with time, but does not repeat, so it is a transient event. The variables events vary repeatedly, rather than periodically, in brightness.

There are two modalities for characterizing light from astronomical objects:

- **spectroscopy**
- **photometry**

Spectroscopy measures the flux as a function of wavelength and is the modern equivalent of using a prism to separate a beam of light into a rainbow of colours. It is a high-precision measurement and it is also the most accurate and reliable tool that enables the classification of astronomical transients and variables. However, spectroscopy is an extremely time-consuming process - with exposure times that are much longer than needed to discover objects with filters on an equivalently sized telescope.

Given the volume of data coming from LSST, obtaining spectroscopic observations for every object is not feasible. An alternative approach is to take an image of the object through different filters (also known as passbands), where each passband selects light within a specific wavelength range. This approach is called photometry, and data obtained through this method are called photometric data; 6 different passabands filters are available for the LSST data, denoted as:

- u ranging from 300-400 nm
- g ranging from 400-600 nm [we can see here]
- r ranging from 500-700 nm [we can see here]
- i ranging from 650-850 nm
- z ranging from 800-950 nm
- y ranging from 950-1050 nm

Classification is performed on these light curves.

1.4.1 Why the problem is so challenging

The observations are sometimes degraded by moonlight, twilight, clouds, and atmospheric effects. These degradations result in larger flux uncertainties, and this information is included in the light-curve data. In the challenge data sets, in addition to light curves, two other pieces of information are provided for each object.

- **redshift**
- **MWEBV**

The redshift is a cosmological measure (denoted by z) that slows down as the rate of arrival of photons is compared to the rate at which they were emitted. An object at z = 1 is 1/8 times longer-lived than an object at z = 0.1. Because higher redshift objects are more distant, the light we receive from them is fainter (and redder), and so we may actually see fewer observations above the signal-to-noise floor.

The effect of redshift on light from a galaxy reaching earth is illustrated in Fig 1.3.

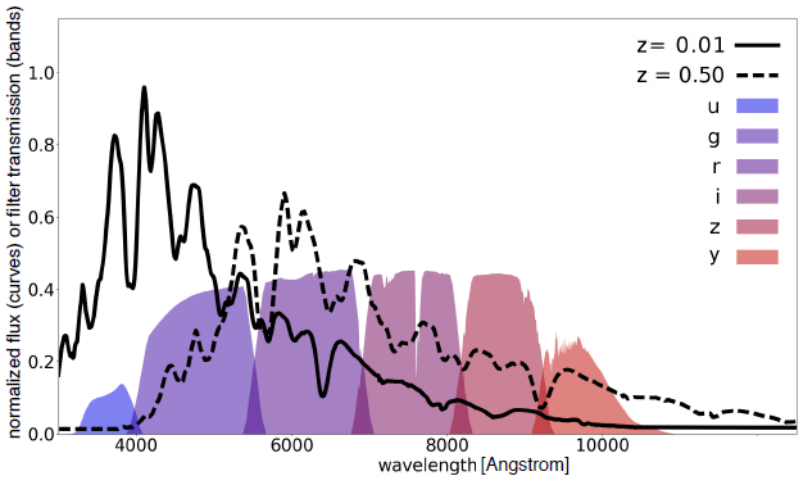


Figure 1.3: The effect of the redshift on the light

The black curve shows a supernova spectrum at a redshift of 0.01, corresponding to a distance of 140 million light years. Visual inspection of the supernova spectrum and the filter efficiencies shows that the maximum flux (spectrum summed over filters) is in the green (g) filter. The dashed curve shows a spectrum from a more distant supernova, corresponding to a redshift of 0.5, or 5.1 billion light years away. The maximum flux has now shifted to the red (r) filter. As the redshift and distance increase, the maximum flux appears in a redder filter: hence the term 'redshift'!.
The MWEBV information is related to extinction of light from our Galaxy, known as the Milky Way. Our light curve measurements are corrected for the atmosphere and telescope transmission.
This absorption is strongest in the ultra-violet u-filter, and weakest in the infrared filters (izy). So that is an astronomical measure of how much redder an object appears compared to a Milky Way without dust. Larger MWEBV values correspond to more Milky Way dust along the line of sight to the objects, making the objects appearing redder. All objects in the dataset are selected to have MWEBV < 3 [1]; to ensure that we are not looking too close to the disc of the Milky Way.

R As references we referred to the official **data-note.pdf** documentation retrieved at: `https://www.kaggle.com/c/PLAsTiCC-2018`

[1] A value MWEBV = 3 means that a large percentage of the light from an object (99% of the light depending on the spectrum of the object) is absorbed by the dust; only the brightest objects could be seen through such high levels of dust

2. Data understanding

2.1 The data

The data we used for the project is available in .csv format, in particular we used 2 data sets:

- metadata: 7848 rows and 12 columns
- time series data: 1421705 rows and 6 columns

In particular for the metadata set we have the following variables:

- *object_id*: the Object ID, unique identifier
- *ra*: right ascension, sky coordinate: longitude, units are degrees
- *decl*: declination, sky coordinate: latitude, units are degrees
- *gal_l*: galactic longitude, units are degrees
- *gal_b*: galactic latitude, units are degrees
- *ddf*: a Boolean flag to identify the object as coming from the DDF survey area[1]
- *hostgal_specz*: the spectroscopic redshift of the source
- *hostgal_photoz*: the photometric redshift of the host galaxy of the astronomical source. It is a far less accurate version of the spectroscopic redshift.
- *hostgal_photoz_err*: The uncertainty on the hostgal_photoz based on LSST survey projections
- *distmod*: The distance (modulus) calculated from the hostgal_photoz since this redshift is given for all objects
- *mwebv*: this 'extinction' of light is a property of the Milky Way (MW) dust along the line of sight to the astronomical source, and is thus a function of the sky coordinates of the source ra, decl.
- *target*: the class of the astronomical source

Instead, for the time series data we have the following variables:

- *object_id*: same of above

[1] DDF: the so-called Deep Drilling Fields are small patches of the sky that will be sampled often to achieve great depth (i.e. to be able to measure the flux from fainter objects). Objects in these DDF patches will have light-curve points that are extremely well determined and therefore have small errors in flux.

- *mjd*: the time in Modified Julian Date (MJD) of the observation.[2]
- *passband*: the specific LSST passband integer, such that u; g; r; i; z; y = 0; 1; 2; 3; 4; 5 in which it was viewed.
- *flux*: the measured flux in the passband of observation as listed in the passband column. The flux is corrected for MWEBV, but for large dust extinctions the uncertainty will be much larger in spite of the correction

  N.B. Due to statistical fluctuations and the way the brightness is estimated, the flux may be negative for few sources, where the true flux is close to zero and positive. Additionally, if the pre-survey image actually contains a flux brighter than its true 'zero', this can lead to a negative flux when the difference is computed.
- *flux_err*: the uncertainty on the measurement of the flux listed above
- *detected*: If detected= 1, the object's brightness is significantly different at the 3σ level relative to the reference template. It is given as a Boolean flag

2.2 Before starting

2.2.1 A background on Time series classification

Time series classification is an important problem in data mining and during time a lot of algorithms have been developed and at the same time, deep learning applications have achieved great performances in such a task. In fact, sequential data can be processed with deep learning which has been shown to be effective in achieving good performances and solving certain tasks. In Figure 2.1, a general framework for time series classification is depicted.

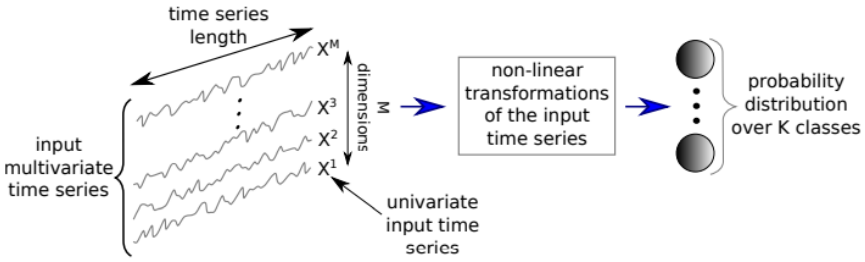


Figure 2.1: DL framework for time series classification

Neural networks are designed to learn the hierarchical representations of the data: many architectures are suitable for time series classification.
In our work we proposed 3 of them:

- LSTM - Autoencoder;
- Convolutional Neural Network;
- GRU-based Neural Network.

We will discuss them in the following sections.

[2]The MJD is a unit of time introduced by the Smithsonian Astrophysical Observatory in 1957 to record the orbit of Sputnik. The MJD is defined to have a starting point of midnight on November 17, 1858. The 25th of September 2018 has an MJD of 58386.

2.3 Data Preparation

It is known that neural networks work better with standardized data so, at first, we proceeded to standardize the time series data. As mentioned previously, each astronomical object has 6 time series, one in each band; so we standardized the fluxes of each object respect to mean and the variance considering all the bands. We took this path because standardization saves the distributional form of the data and also because it is more robust when it comes to outliers (i.e. a point that is an outlier before the standardization will continue to be an outlier after it).

2.4 Explorative Analysis

A brief exploratory analysis was conducted in order to have a better understanding of the data in hand; moreover, domain knowledge was acquired after meetings with Professor Giuseppe Longo. The following picture [Figure 2.2] represents the correlation matrix between the variables of the 'metadata' dataset.

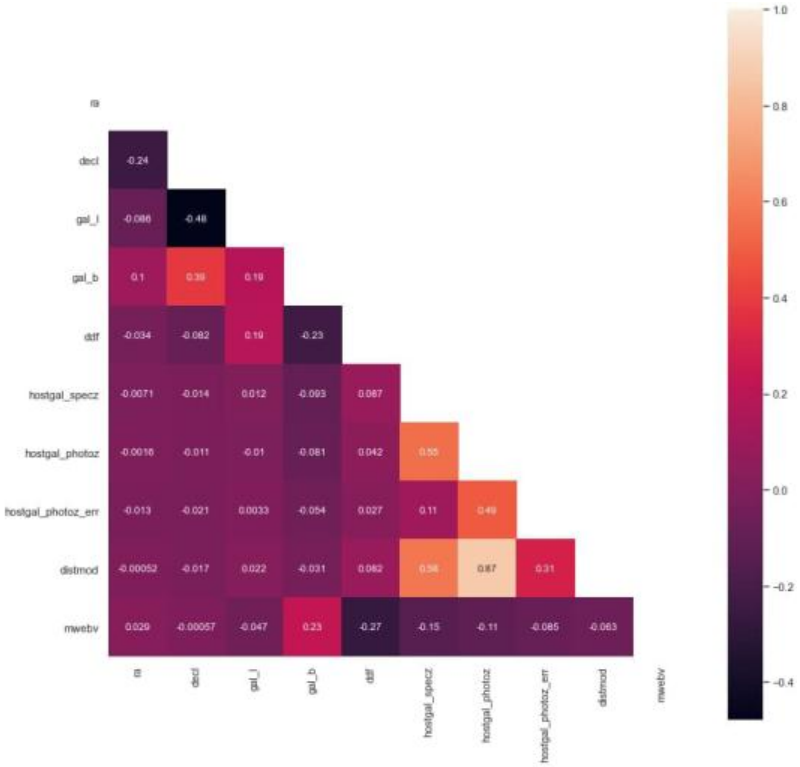
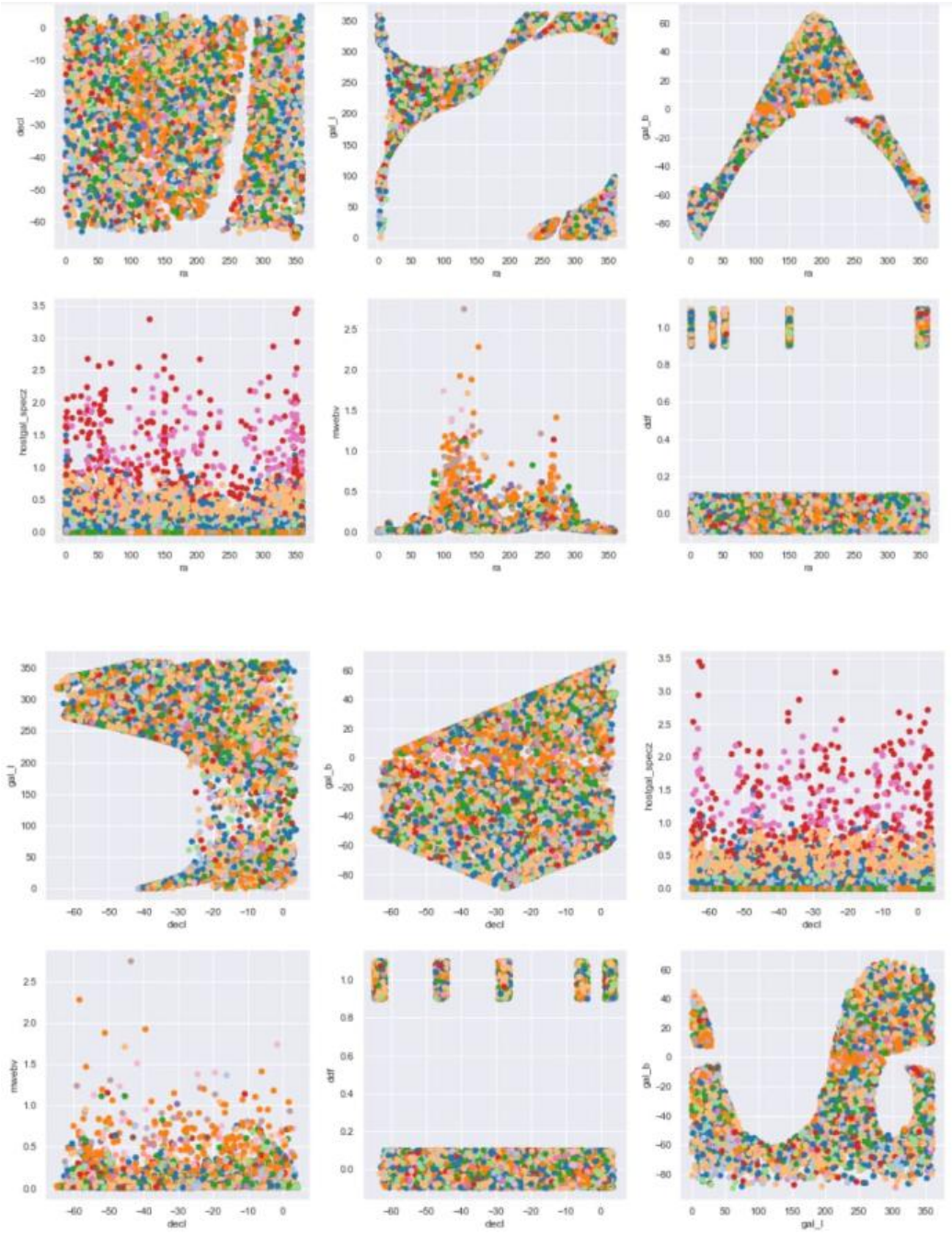


Figure 2.2: Matrix of correlations among variables

The only variables that report a correlation greater than 0.5 are:

- hostgal_specz - distmod: this correlation makes sense from a physical point of view
- hostgal_specz - hostgal_photz: these are the measures of the same quantity, so the correlation is legitimate
- hostgal_photoz - distmod: this correlation makes sense from a physical point of view

These results, though, do not provide enough information about significant variables that can help in the classification and since we were unable to get any information, we tried a different path using visual analysis. These are the plots that came out from this new approach:



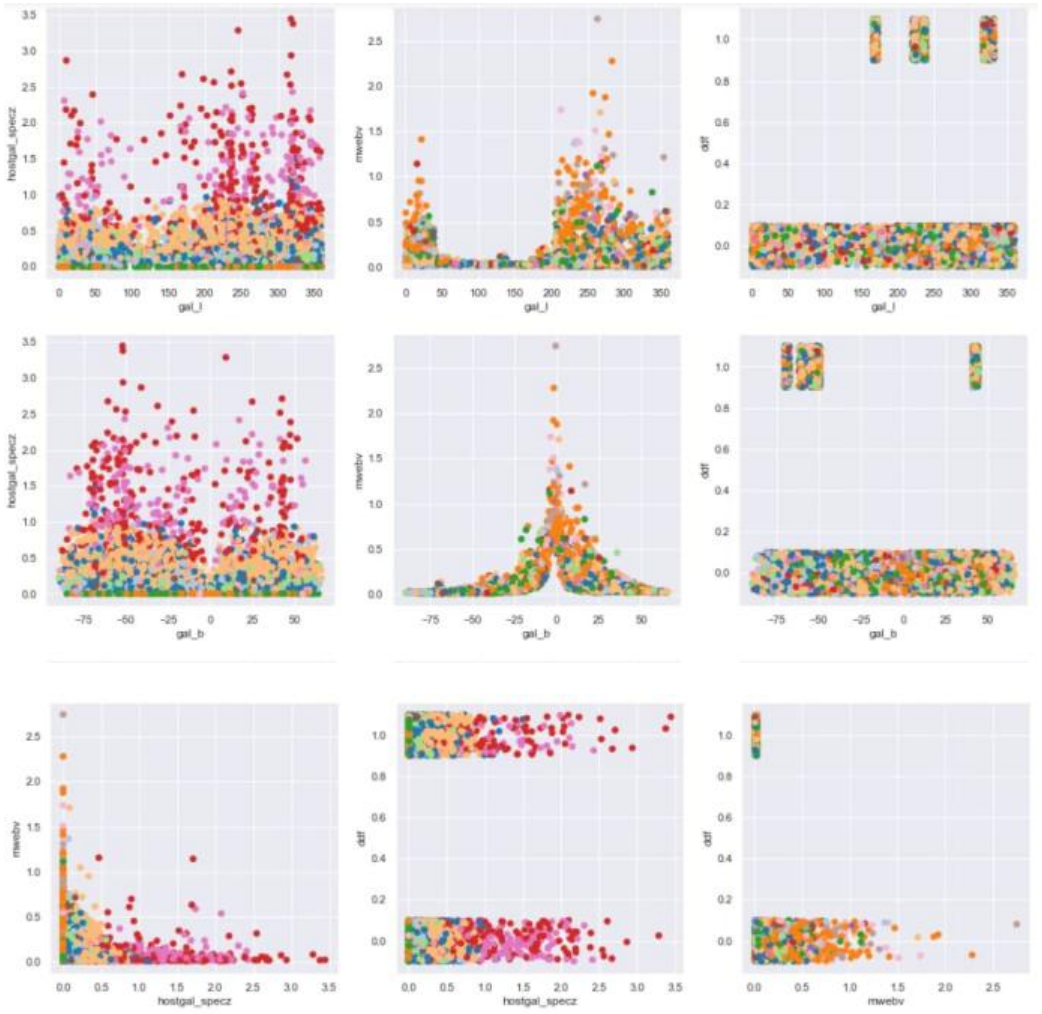


Figure 2.3: The colours of the points represent the class to which the object belongs, meanwhile on the axes we have the variables.

From Figure 2.3 it is evident that some of the features seem to be significant, in particular for:

- hostgal_specz
- MWEBV

It is evident that moving along the direction of the above variables we can observe a separation of some clusters of points.

R The observed results are significant according to our domain knowledge, since the spectroscopic redshift is a high precision measure of the distance of an object, so we expect it to have a better discriminative power on certain groups of objects. Same thing for the MWEBV measure, in fact in the last plot of Figure 2.3 is evident that there is a separation respect to the direction one decide to move for the points.

2.4.1 Features importance

The XGBoost library provides a powerful tool to perform gradient boosted trees used to conduct a features importance analysis to get statistical information from the available variables.

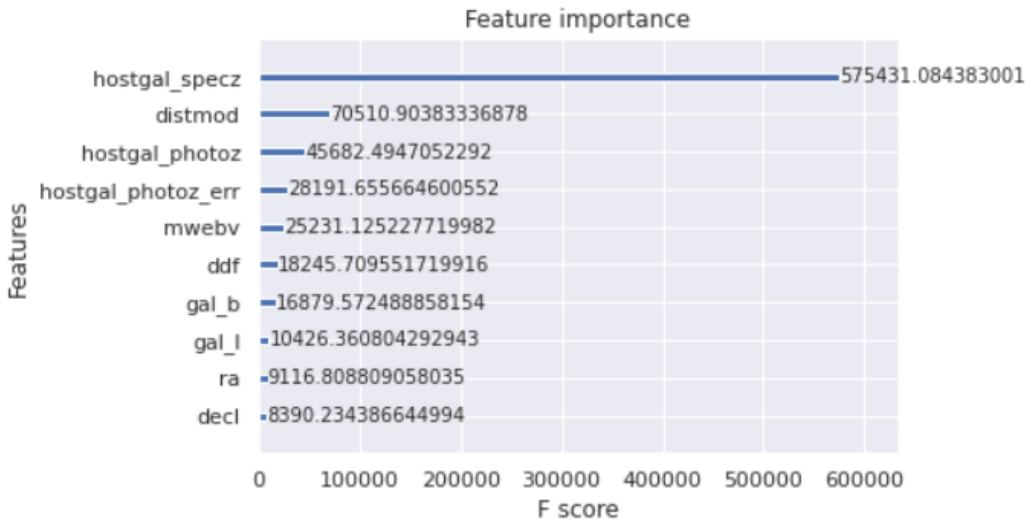


Figure 2.4: Illustrations of the features importance respect to the XGBoost analysis.

XGBoost uses three different methods to choose the importance type of each variable:

- **weight**: it is the total number of times that a certain feature was used to split the data across all trees. It is the default method;
- **gain**: it is the average loss reduction gained when using this feature for splitting in trees;
- **total_gain**: it is the total gain across all features each time they are is used in the splitting process. [3]

From the above, we decided to use the last one method, that seemed to be more appropriate because we were interested in the total gain of information needed to reduce the loss, whereas to train the model considering a multi-classification task we used the following 'multiclass-logloss' as suggested by the official XGBoost documentation.[4]

$$L_{log}(y,p) = -(y * log(p) + (1-y) * log(1-p))$$

At each step of the boosting tree we used a Random Forest with 10 parallel trees, for a total of 100 iteration. We know that boosted tree is influenced by previously grown trees. Furthermore, about the decision of the Random Forest, at each split in the tree the algorithm is not allowed to consider a majority of the available variables but indeed only the square root of the numbers of the predictors. Finally, knowing that trees grown too much may be prone to overfitting we decided to not go deep in

[3] Information gain is based on the concept of entropy and information content. Entropy is defined as: $H(T) = -\sum_{i=1}^{j} p_i log_2(pi)$ and so $IG(T) = H(T)_{parent} - H(T|a)_{children}$ where a is the splitting variable

[4] https://xgboost.readthedocs.io/en/latest/index.html

the full trees.

Apart from the theoretical background of the XGBoost, we can observe that the most important feature is *hostgal_specz*. Note that from the domain knowledge we know that *distmod* and *hostgal_photoz* (and hence the *hostgal_photoz_err*) express the same cosmological concept so we can use only *hostgal_specz*. According to the explorative analysis *gal_l*, *gal_b*, *decl* and *ra* are not significant features and thus they will not be used in some of the models that we will train.

2.4.2 Further insights

By observing the data we realised that each band had a different number of points and that the distribution of the target is not uniform, therefore we wanted to understand the overall situation across all the objects. Moreover, the
In order to do that, we plotted the following charts.

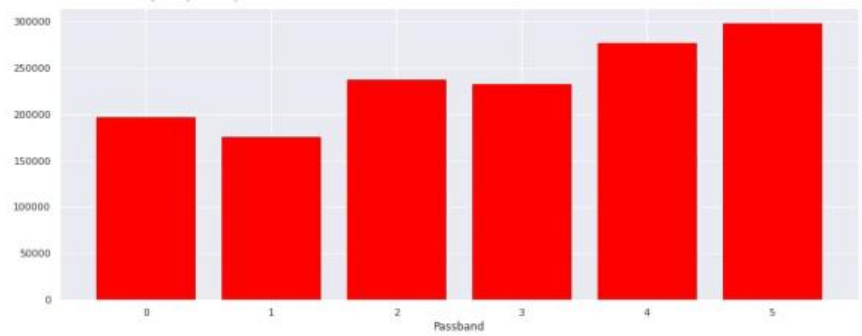


Figure 2.5: Overall number of points per band.

From Figure 2.5 it is possible to notice that the number of points in each band is significantly different, which implies the need of a diverse preprocessing of the data in each of them.

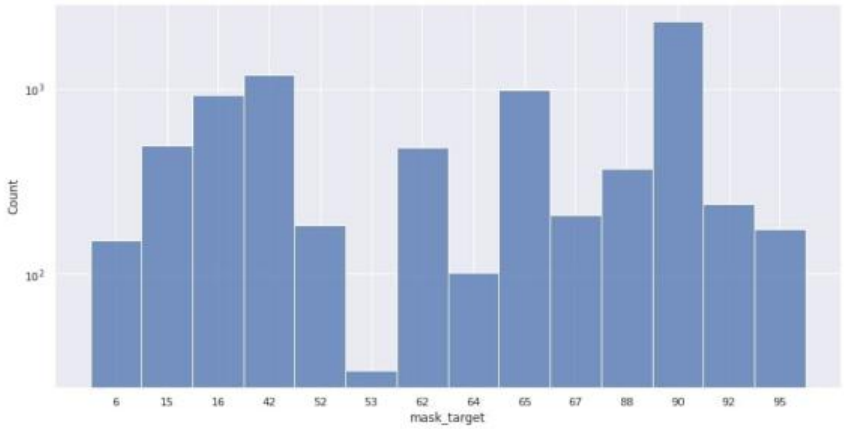


Figure 2.6: Overall number of objects per target.

From Figure 2.6 it is evident that the dataset is unbalanced along the target variable. The count is represented in log scale to better appreciate and compare the numerosity of objects in each target. As the plot shows the target 53 is the one with less objects and so we expect a lot of errors in the classification of objects belonging to this target. Moreover, considering the log scale used, we need to remember that bars that are half of others present ten times less objects than the others.

3. Modeling

3.1 Recurrent Neural Networks (RNNs)

Recurrent Neural networks represents the perfect candidate for sequential data such as time series. They are formed by the following elements:

1. Recurrent neurons and layers;
2. Memory cells.

In a feedforward neural network (for example we can think of a basic MLP architecture) the inputs flow from the left to the right of the network, meanwhile in a recurrent network the inputs not only flow in one direction through neurons but they also have back connections, so a neuron receives an input, produce an output and then send it back to itself. At each time step, the neuron receives the inputs $\mathbf{x}_{(t)}$ as well as its own output from the previous time step, $\mathbf{y}_{(t-1)}$. To figure out what just said refers to the following figure 3.1:

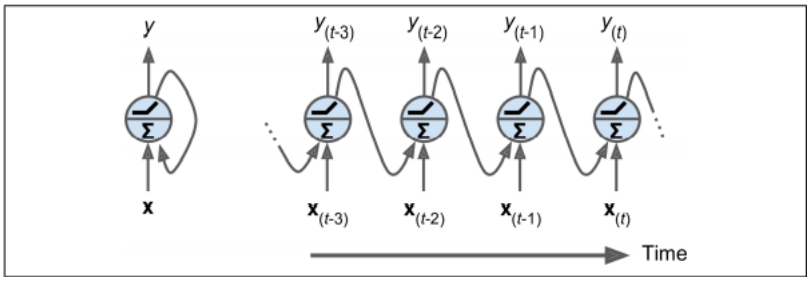


Figure 3.1: Recurrent Neuron unrolled through time

At each time step t, every neuron receives both the input vector $\mathbf{x}_{(t)}$ and the output vector from the previous time step $\mathbf{y}_{(t-1)}$. Considering the whole recurrent layer, it has two weight matrices $\mathbf{W}_x$ and $\mathbf{W}_y$ and the output vector of the recurrent layer can be computed according to the following formula:

$$\mathbf{y}_{(t)}=\phi(\mathbf{W}_x^T\cdot\mathbf{x}_{(t)}+\mathbf{W}_y^T\cdot\mathbf{y}_{(t-1)}+\mathbf{b})$$

Since the output of a recurrent neuron at time step t is a function of all the inputs from previous time steps, we could say it has a form of *memory*, and so the part of the RNN that preserve information across time steps is called *memory cell*. A cell's state at time step t, denoted by $\mathbf{h}_{(t)}$, is a function of some inputs at that time step and its state at the previous time step: $\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$. Its output at time step t, denoted $\mathbf{y}_{(t)}$, is also a function of the previous state and the current inputs. In order to learn from long sequences we need the network to run over many time steps, meaning that the RNN will slowly begin to forget about the further previous data points. LSTM cells tackle this problem.

3.1.1 LSTM cells

As we know, deep neural networks may suffer from gradient problems. Moreover, when a RNN processes a long sequence, it will gradually forget the first data points of the sequence. Due to the transformations that the data goes through when traversing a RNN, some information is lost at each time step. After a while, the RNN's state contains virtually no trace of the first points. The RNN proposes the LSTM cell, also known as *Long Short-Term Memory* where long-term dependencies in the data are better detected. From the following figure 3.2 we can understand how the LSTM cell works:

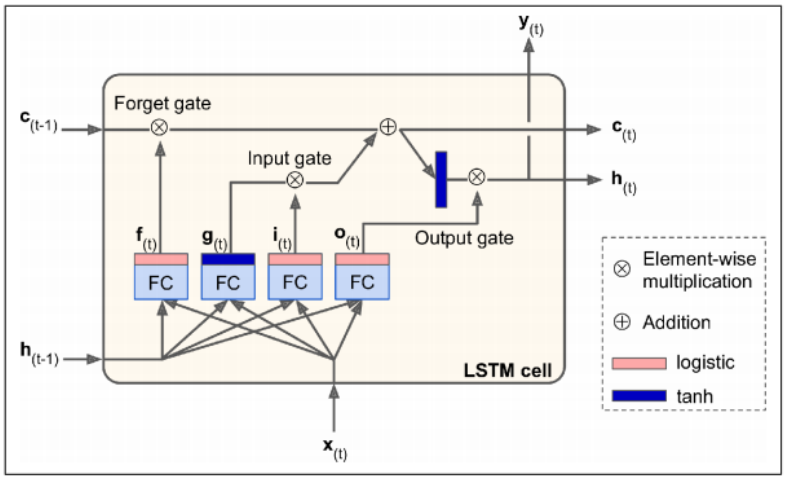


Figure 3.2: LSTM cell

The state of the LSTM is split into two vectors:

- $h_{(t)}$**: short-term state**
- $c_{(t)}$**: long-term state**

The network can learn what to store in the long-term state, what to throw away, and what to read from it. As the long-term state $\mathbf{c}_{(t-1)}$ traverses the network from left to right, it is clear that it first goes through a *forget gate*, dropping some memories, and then it adds some new memories via the addition operation. So, at each time step, some memories are dropped and some memories are added. Moreover, after the addition operation, the long-term state is copied and passed through the *tanh* function, and then the result is filtered by the output gate. This process produces the short-term state $\mathbf{h}_{(t)}$. Where do the memories come from? How do the gates work? The current input vector $\mathbf{x}_{(t)}$ and the previous short-term state $\mathbf{h}_{(t-1)}$ are fed to four different fully connected layers. They all serve a

different purpose:

- the main layer is the one that outputs $\mathbf{g}_{(t)}$, it has the role of analazying the current inputs $\mathbf{x}_{(t)}$ and the previous state $\mathbf{h}_{(t-1)}$. The important parts of its output are stored in the long-term state.
- The three other layers are *gate controllers*. Since they use the logistic activation function, their outputs range from 0 to 1.

Respectively, we have:

- The *forget gate* (indicated by $\mathbf{f}_{(t)}$): that controls which parts of the long-term state should be cancelled;
- The *input gate* (indicated by $\mathbf{i}_{(t)}$): that controls which parts of $\mathbf{g}_{(t)}$ should be added to the long-term state;
- The *output gate* (indicated by $\mathbf{o}_{(t)}$): that controls which parts of the long-term state should be read and the output at this time step, both to $\mathbf{h}_{(t)}$ and to $\mathbf{y}_{(t)}$.

So in short, an LSTM cell can learn to recognize an important input, store it in the long-term state, preserve it for as long as it is needed, and extract it whenever it is needed.

3.1.2 GRU cells

The GRU cell, also known as *Gated Recurrent Unit*, is a simplified version of the LSTM cell, and it seems to perform just as well.

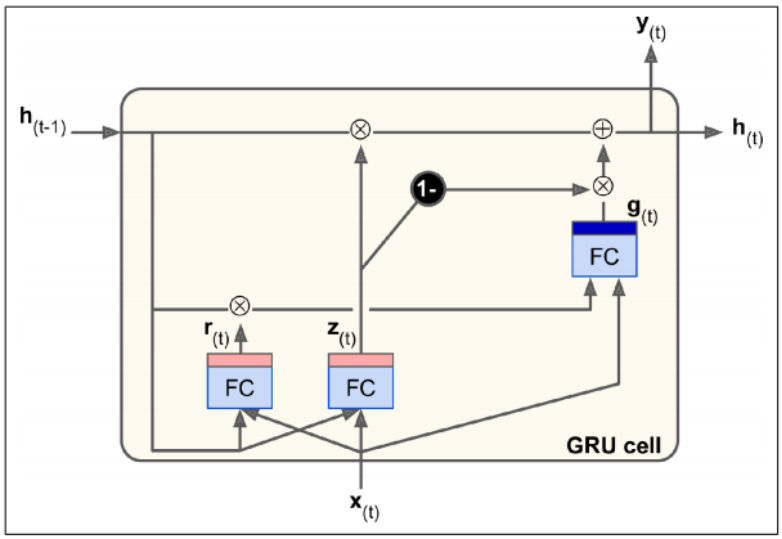


Figure 3.3: GRU cell

Here both state vectors are merged into a single vector $\mathbf{h}_{(t)}$.
Moreover, a single gate controller $\mathbf{z}_{(t)}$ controls both the forget gate and the input gate. If the gate controller outputs a 1, the forget gate is open (= 1) and the input gate is closed (1 – 1 = 0). If it outputs a 0, the opposite happens. Finally, the full state vector is output at every time step. However, there is a new gate controller $\mathbf{r}_{(t)}$ that controls which part of the previous state will be shown to the main layer ($\mathbf{g}_{(t)}$).

3.2 LSTM Autoencoder

We built an ad-hoc architecture to manage uneven sampling and missing data. We handled missing data marking them with a flag, and forcing the LSTM layer to skip computation on them. We faced the problem of uneven sampling of time series by introducing the feature Δt that allows the architecture to take into account a non-uniform sampling. The architecture [1] proposed is presented in the following figure.

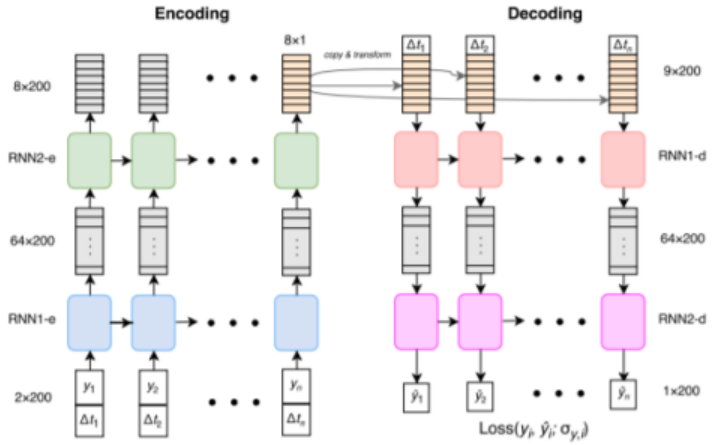


Figure 3.4: Architecture of the LSTM Autoencoder model

This network uses two RNN layers (specifically, bidirectional gated recurrent units (GRU)) of size 64 for encoding and two for decoding, with a feature embedding size of 20. The encoder takes as inputs the measurement values as well the sampling times (more specifically, the differences between sampling times); the sequence is processed by a hidden recurrent layer to produce a new sequence, which can then be used as the input to another hidden recurrent layer, etc. The fixed-length embedding is constructed by passing the output of the last recurrent layer into a single fully-connected layer with a linear activation function and the desired output size. The decoder first repeats the fixed-length embedding n_T times, where n_T is the length of the desired output sequence, and then appends the sampling time differences to the corresponding elements of the resulting vector sequence. The sampling times are passed to both the encoder and decoder; the feature vector characterizes the functional form of the signal, but the sampling times are needed to determine the points at which that function should be evaluated. The remainder of the decoder network is another series of recurrent layers, with a final linear layer to generate the output sequence. We also apply 25% dropout between recurrent layers, which is omitted from the figure for simplicity. In our model we take the number and size of recurrent layers in the encoder and decoder modules to be equal, but in general the two components are entirely distinct and don't need to share any architectural similarities.

[1] https://www.nature.com/articles/s41550-017-0321-z

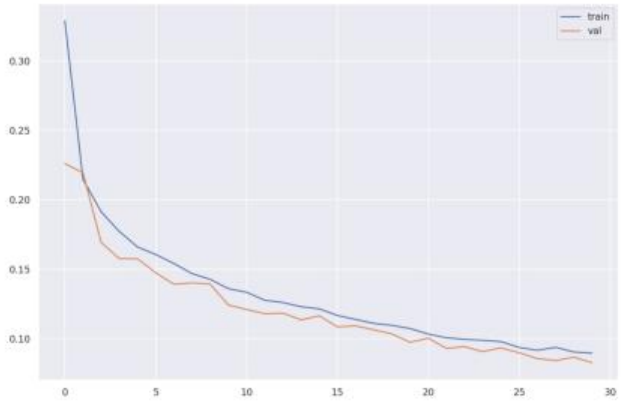


Figure 3.5: Number of epochs on the x-axis versus losses on the y-axis

For our project we trained the LSTM Autoencoder for 30 epochs, obtaining a validation loss of 0.0827 (figure 3.5). With the autoencoder, the target is equal to the input vector itself, so the goal of training the autoencoder is a well reconstruction of the input signal. Since we have a lot of marked missing values and we do not want the architecture to reconstruct the signal with this lack of information, we wrote a custom loss that calculates the mean squared error only on available points. The output of the only encoder part is a matrix for each astronomical object, of dimensions *(number of reduced dimensions) x (number of bands)*.
The following figure shows the signal rebuilt by the autoencoder:

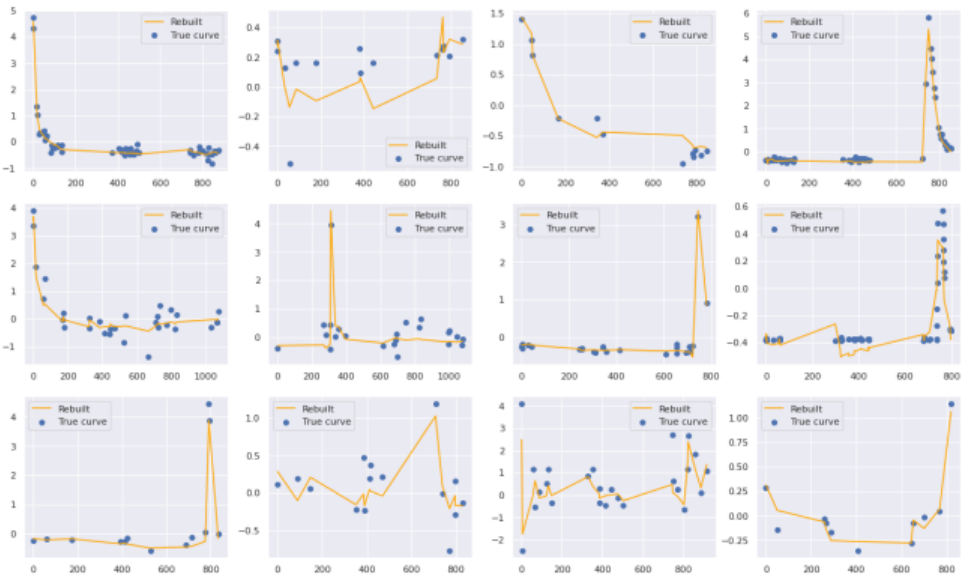


Figure 3.6: Light curves rebuilt by the autoencoder

3.3 The way of the splines

Regarding the problem of uneven sampled time series we decided to follow an alternative method to the autoencoder, considering the signal rebuilt by the splines.

A spline function of order **n** is a piecewise polynomial function of **n-1** degrees in a variable x. The places where the pieces meet are known as knots. The key property of spline functions is that they and their derivatives may be continuous, depending on the multiplicities of the knots.

B-splines of order n are basis functions for spline functions of the same order defined over the same knots, meaning that all possible spline functions can be built from a linear combination of B-splines, and there is only one unique combination for each spline function.

When facing the modelling of data, the underlying function is other than linear, in fact situations where non linearity is involved are very common. Often, the nature of the function which describes the data is unknown and what we try is to model it with non linear functions. A well known solution is to write it as linear combination of basis splines, which allows us to move beyond linearity.
The algorithm for the construction of basis splines (Coox - De Boor) it's described in the following section.

3.3.1 Construction of basis splines

Step 1

Define an augmented knots sequence:
$\xi^* = (\xi_{-p}, ..., \xi_0, \xi, \xi_{k+1}, ..., \xi_{k+p+1})$, where p is the degree of the spline and K is the number of internal knots, so that k+p+1 equals to the total degree of freedom related to the method.

Step 2

Build the first order base for the observed data (p=0), which gives the fundamental for a step-function spline. For $i = 0, ..., k$, let

$$B_{i,0}(x) = \begin{cases} 1 & x \quad \in \quad [\xi_i, \xi_{i+1}) \\ 0 & otherwise \end{cases} \tag{3.1}$$

Where $B_{i,0}(x) = 0$ if $\xi_i = \xi_{i+1}$.

Step 3

The i-th B-spline basis function of degree j, $j = 1, ..., p$ is given by:

$$B_{i,j}(x) = \frac{x - \xi_i}{\xi_{i+j} - \xi_i} * B_{i,j-1} + \frac{\xi_{i+j+1} - x}{\xi_{i+j+1} - \xi_{i+1}} * B_{i+1,j-1}$$

for $i = -p, .., k + p - j$.

The result can be seen in the following figure 3.7, showing a strong non linear composition of the basis functions.

R For each x value, the summed components of the basis is equal to 1.

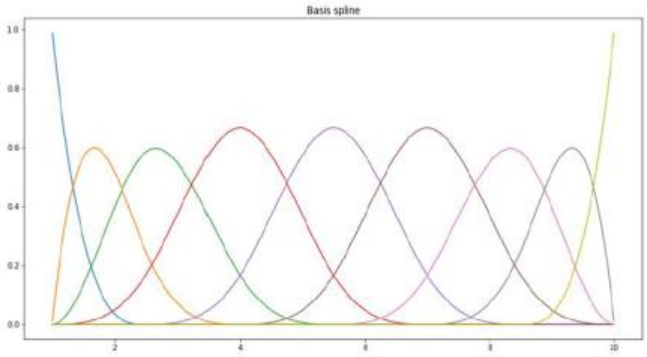


Figure 3.7: Example output of basis spline of degree 3 with 5 equally spaced internal knots.

3.3.2 Spline Modeling

In our analysis we tried to experiment a different way of managing the data, in particular by regularizing the light curves through the B-splines methodology. Although the basis are built by the disposition of the equid-spaced knots, given our lack of sample points in specific regions within the time series, we actually used a density distributed knots positioning: the more concentrated and rapidly varying the data points, lesser distanced the knots positions, as the figure 3.8 shows.

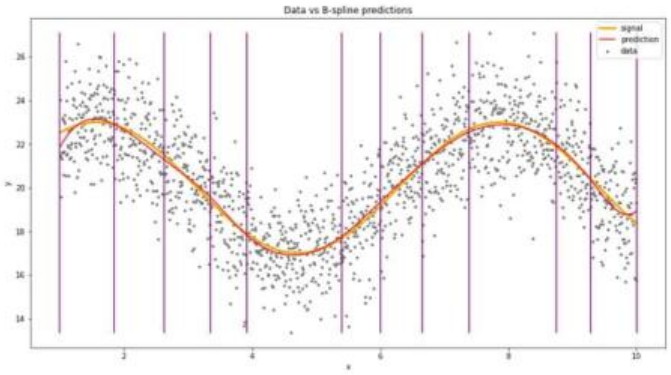


Figure 3.8: B-splines fit with not-equally spaced knots

In order to smooth distributions, the spline methodologies use some biases while describing the data, indeed through the introduction of an a-priori target variable distribution (in our case Gaussian) and an order degree of the curve. This curve is supposed to catch the true signal of a series, despite the statistical fluctuations.
The underlying idea was to re-sample the data with the purpose of a better evaluation of the correlations among the bands (now completely filled). Looking at the data we realized that the light curves at different passband levels have different amount of information. Specifically, the low wavelength bands have a lesser number of points in respect to the other bands and therefore, fitting a B-spline on the data with a lower curve degree and a lower number of knots seems to be more indicated for those. Thus, degrees and knots number have to increase along the bands as we can notice from the following table:

band	number of knots (K)	order of the spline (ord)
0	6	5
1	6	4
2	8	6
3	10	7
4	12	8
5	12	9

Table 3.1: Curve degrees and knot number used in our analysis.

Trying to predict the signal underlying a time series without considering this lack of information can lead to an almost sure overfitting of the signal itself.
To reduce the problematic as much as possible, we proposed a method based on the adoption of masking layers, as depicted in the following figure:

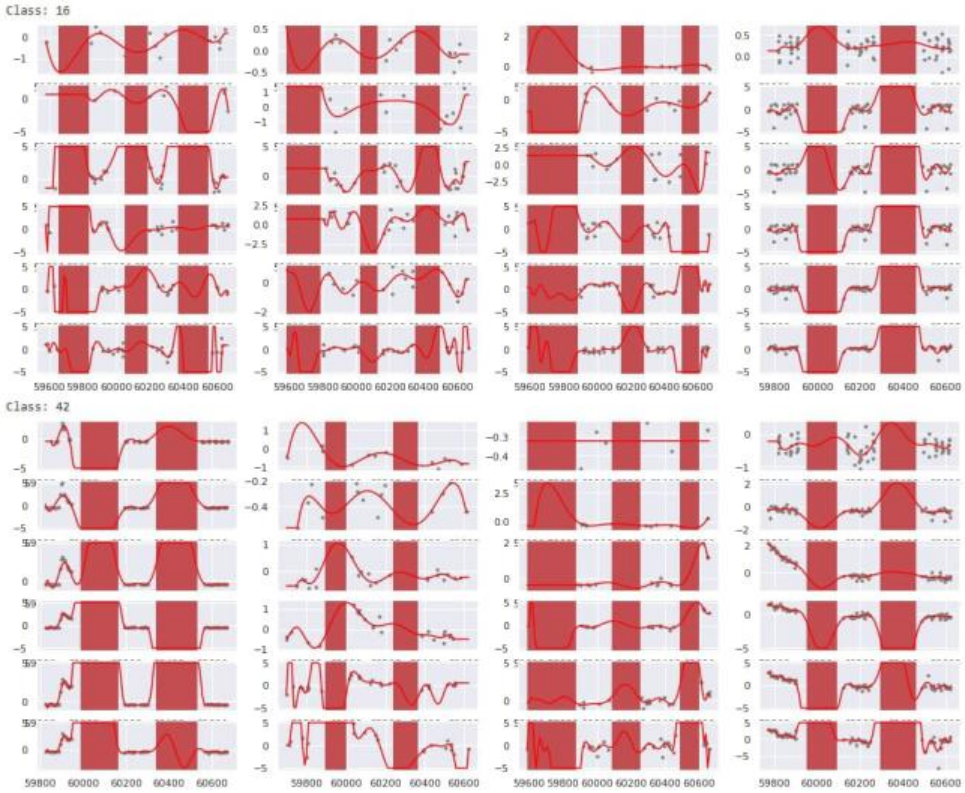


Figure 3.9: Fitted light curves signals by the splines. The red rectangles represent areas where the are no points and the splines are forced to not fit that regions.

Instead of generating a fixed length smoothed version of the real time series, we assigned to the 'unfaithful' time values a certain number (-10.0 for instance) despite the predicted value from the B-splines. The unfaithfulness of a time value is given by a score system depending on the positions and concentration of the light curve points and it was used in order to avoid the issues caused by inference on regions in which no data points are present. Basically, at each equally distanced value within an interval of time has been assigned the minimum of the distances between that value and every other time value of the true distribution in a specific band. Then, we reduced all these partial scores referring to each band in a weighted sum:

$$Score(x) = \sum_{band} (w_{band} \cdot p_{score}^{(band)}(x))$$

Where $p_{score}^{(band)}(x) = min\{d(x,y) \mid y \in support\ of\ light\ curve \cap support\ of\ the\ specific\ band\}$.

In such way we were able to differentiate the bands in respect to their overall numerosity (figure 2.5), thus setting the latters as our weights.
In figure 3.10 we can look at some examples of the score function applied to the time ranges of three light curves.

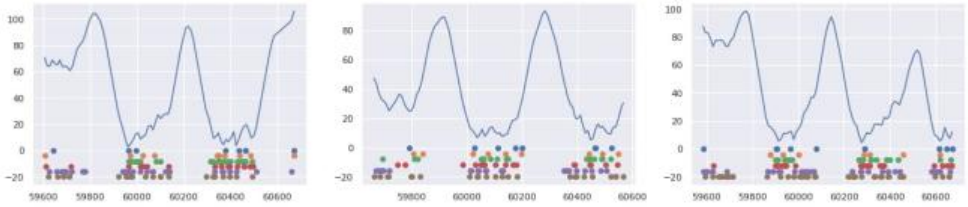


Figure 3.10: The figure shows the time value distribution for the bands (in different colors) at the bottom and a blue line representing the score function applied on every point belonging to the broadest time interval.

As we can see, higher values of the score function correspond to more thinned out regions. Fixing a threshold (figure 3.11) we are able to construct the masked interval mentioned above, that is time ranges that will not be considered throughout the entire learning process.

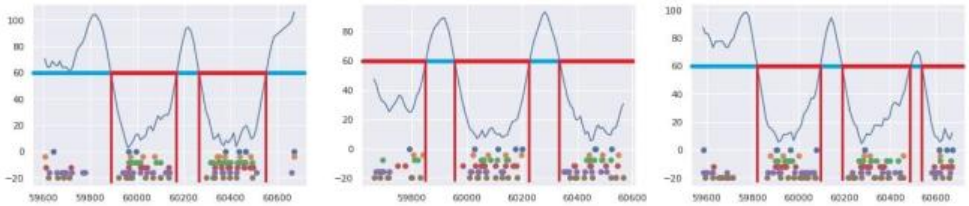


Figure 3.11: Red horizontal lines represent regions with higher density of points, meanwhile blue lines corresponds to those that will be masked.

3.4 Convolutional Neural Networks(CNNs)

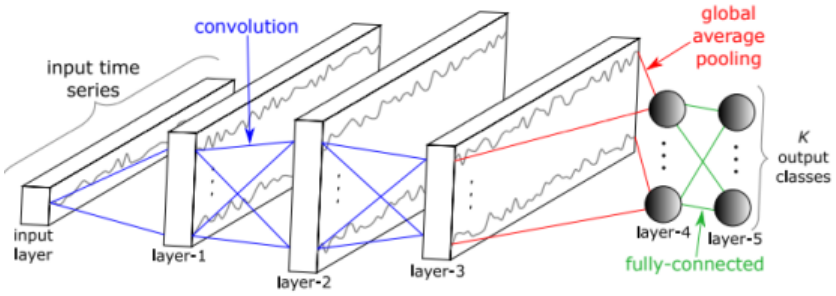


Figure 3.12: General architecture of a Convolutional Neural Network

3.4.1 Introductory notes

Although CNN are famous to be the must to use architecture in image processing and computer vision applications, in the recent years they not only have reached human performances in image recognition tasks but at the same time they have reached a lot of successful applications in many other tasks, i.e. time series classification. Roughly speaking, convolution can be seen as the sliding of a filter on the time series. During the process of convolution, the filter must be one-dimensional and so, the application of it can be interpreted as generic non-linear transformation of the time series (after the filter, an activation function is applied). The process of convolution with the filter (sometimes referred to as *kernel*) is interpreted as a moving average with a sliding window having the same length of the filter dimension. A general form of the process of convolution is depicted in the following formula:

$$C_t = f(\omega \cdot X_{[t-l/2:t+l/2]} + b) | \forall t \in \left[\frac{l}{2}, T - \frac{l}{2}\right]$$

where:

- C is the result of the convolution;
- f is a general non-linear function (i.e. might be the ReLU function);
- $\cdot$ is the dot product;
- X is an univariate time series;
- T is the length of the time series;
- ω is a filter of length l;
- b is the bias parameter.

From this general definition it is clear that applying several filters on an input time series would mean to learn multiple significant features useful for the classification task.

An interesting property of the CNNs is called *weight sharing* that allows the CNN to learn filters that are invariant across the time dimension; the values of the filter ω are learned automatically from the network, in particular a *pooling* operation is performed before the final fully - connected network classifier.

The pooling operation can be:

- local and global
- average and max

Local pooling such as average or max pooling takes an input time series and reduces its length T by aggregating over a sliding window of the time series. With a global pooling operation, the time series will be aggregated over the whole time dimension resulting in a single real value. In other words, this is similar to applying a local pooling with a sliding window's length equal to the length of the input time series. Usually a global aggregation is adopted to reduce drastically the number of parameters in a model thus decreasing the risk of overfitting. Finally, batch normalization operations are performed in order to help the network to converge quickly.

> Recall that CNN is viewed as a feature extractor, the task of the classification is, as already said, performed by the final *softmax* layers of the MLP architecture.

3.5 Our architectures

Being in compliance with the challenge we decide to monitor and train our architectures considering the loss function given within the challenge context:

$$L = -\frac{\sum_{j=1}^{m} w_j \cdot \sum_{i=1}^{n} \frac{1}{n_j} \cdot \tau_{ij} \cdot log(p_{ij})}{\sum_{j=1}^{m} w_j}$$

. Where:

- w_j are the weights used in the context; they are hidden to the participants so we estimated them with normalized frequencies;
- n_j represents the number of points;
- τ_{ij} is a dummy variable and it is equal to 1 if i-th object comes from the j-th class;
- p_{ij} represents the probability that the i-th object belongs to the j-th class.

Filtering along the bands allows us to extract features not only along time, but also considering the joined information between the bands. Moreover, we have added to the metadata two more variables *mean* and *std* of the time series, since they have been extracted during the preprocessing phase in order to standardize them. Then, we tried again to evaluate the features importance through the XGBoost methodology, keeping the previous metric (figure 3.13).

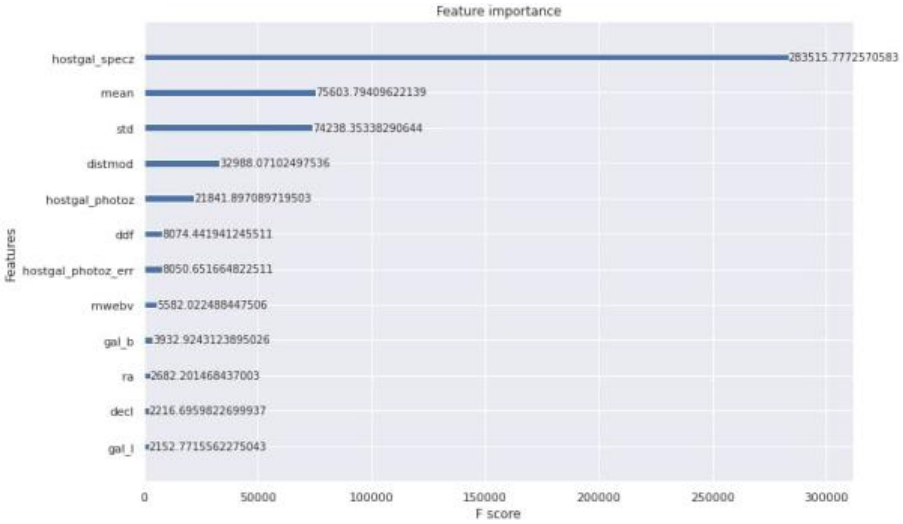


Figure 3.13: Structure of the Exotic Architectures

Eventually, we noticed that these just mentioned variables discriminate the target, as well as the other features *hostagal_specz* and *MWEBV*. Therefore, in some further stage of the training they will be the only variables considered. During the the modeling phase, we used multiple configurations of the available data given by the cartesian product of the following sets:

- 1. Complete dataset: all data points are going to be used during the training process;
  2. Incomplete dataset: just a portion of the entire set of data, in particular those objects having at least 10 data points in each band, can be used in the next phase.

- 1. Autoencoder reduction: training and test data points are going to be encoded by the autoencoder;
  2. Spline re-sampling: training and test data points will be sampled from fitted B-splines.

- 1. CNN architecture: CNN-based approach;
  2. Exotic architecture: GRU-based approach.

In both the architectures, a final MLP has been adopted together with a softmax activation function which allows us to perform the classification task. Despite the initial different sections of these models, they share a concatenation layer that ties up the normalized metadata together with the output of their characteristic phases (convolution for the first and recurrency for the latter).

3.5.1 CNN application

We decided to apply a CNN architecture (figure 3.14) to take into account the nature of the multivariate behaviour of the time series. The input of the CNN is a matrix for each astronomical object of the following dimensions:

- number of time steps
- number of bands

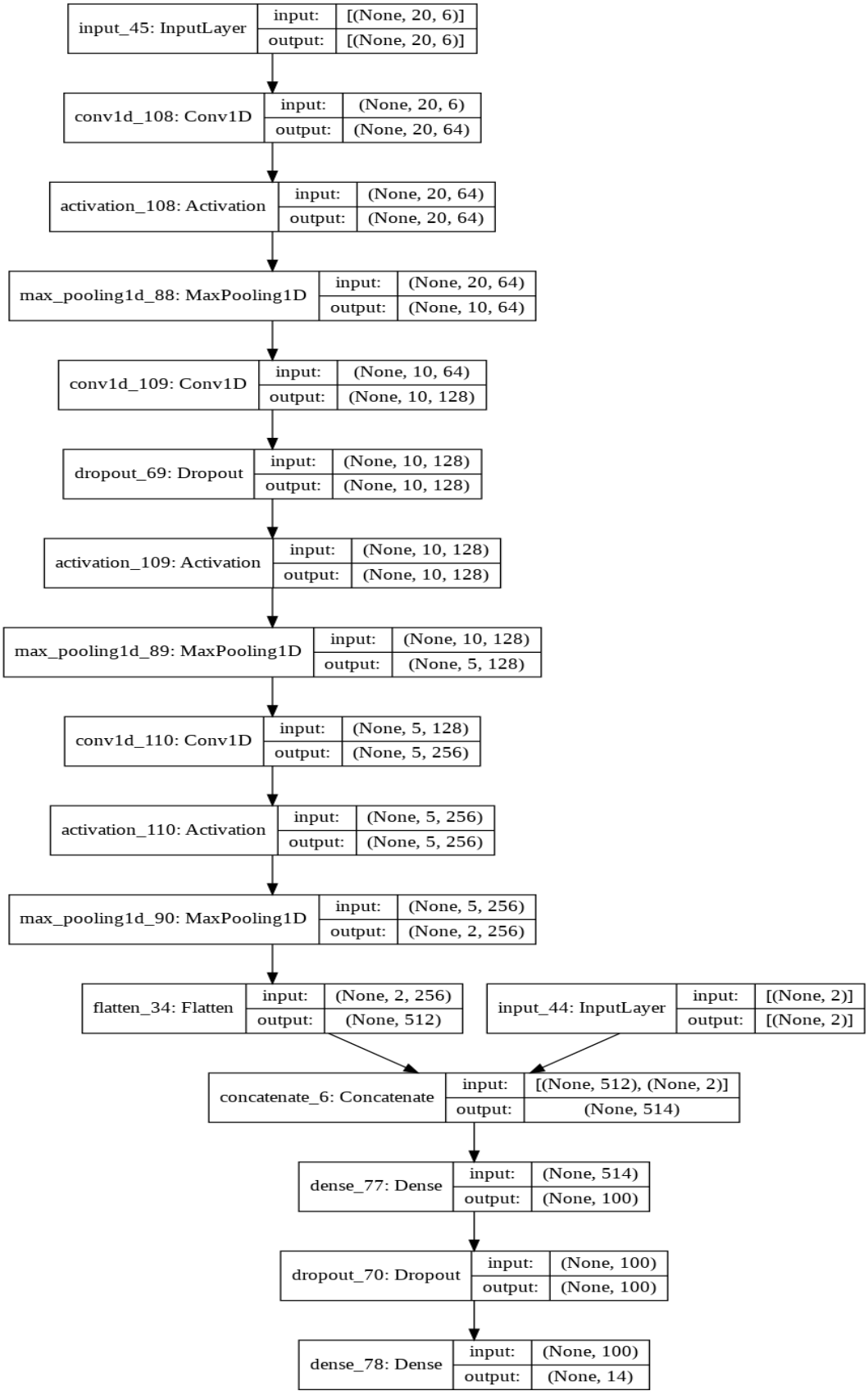


Figure 3.14: Example of a CNN Architecture instance based on non-optimized hyperparameters

3.5.2 Exotic application

While the CNNs are well suited for understanding the nature of the data throughout correlations between the bands, the GRU-based architectures, on the other hand, try to find the dependences of each point among the time series and thus, the global traits of them. Figure 3.15 shows an example of this kind of architecture.

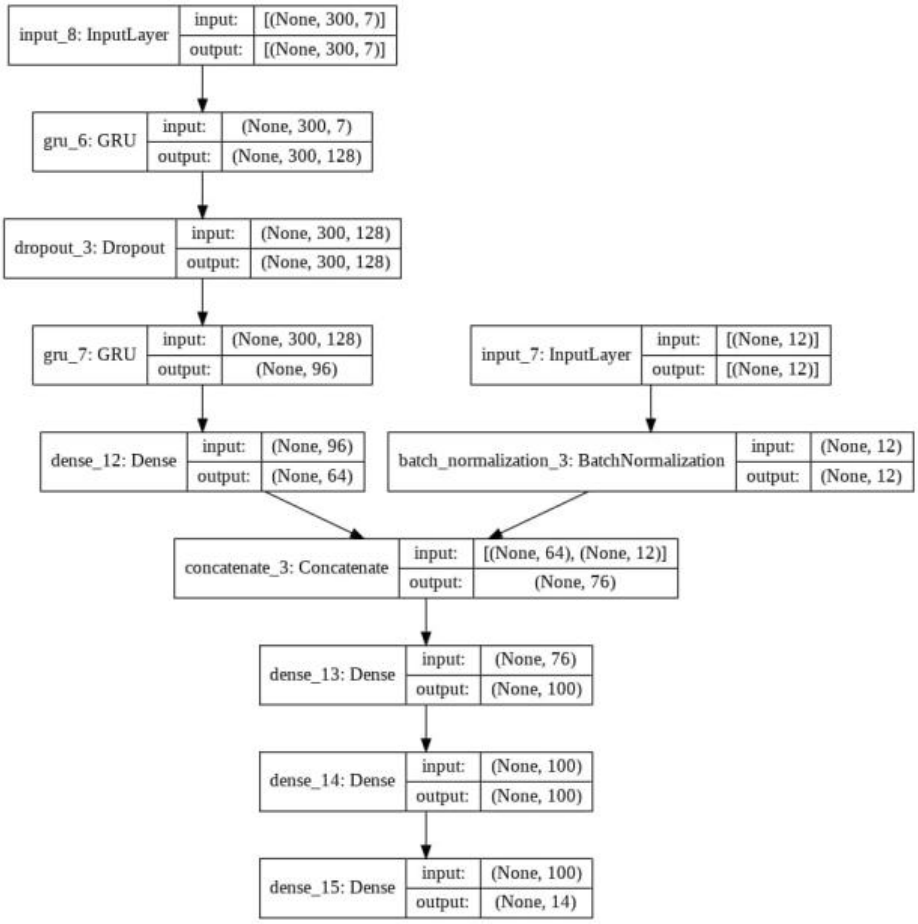


Figure 3.15: Example of an Exotic Architecture instance based on non-optimized hyperparameters

4. Conclusions

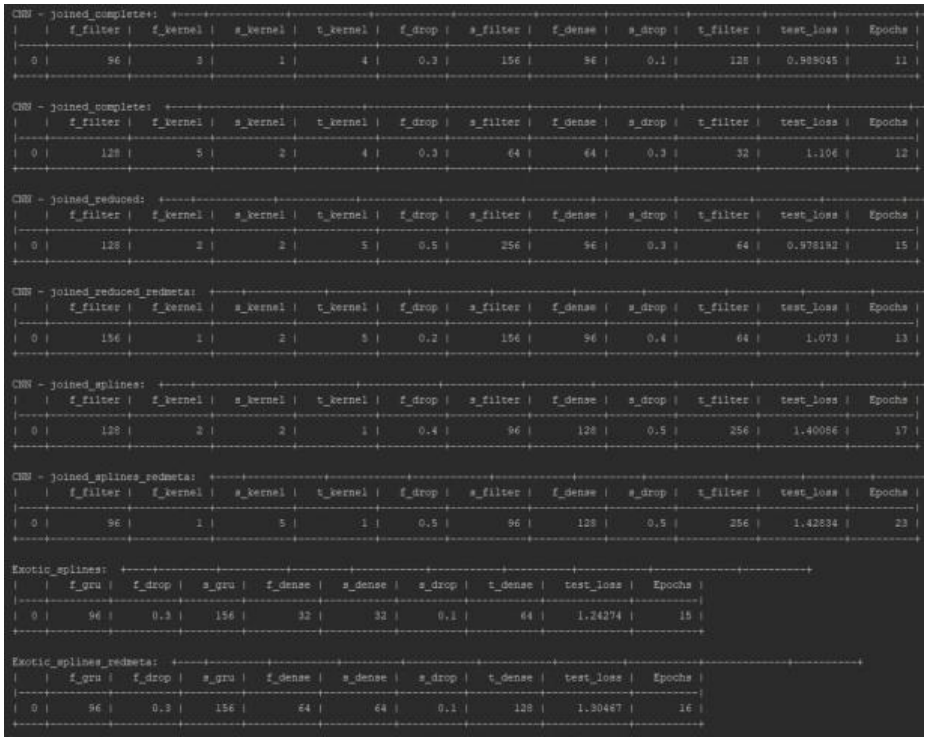
Let us now compare the results of the 3 macro-architectures that we developed, which are:

- *AutoEncoder + CNN*
- *Spline + CNN*
- *Stacked GRUs*

Each of these architectures has been trained on different data as follows:

- In the *CNN - joined_complete+* architecture both autoencoder and CNN have been trained on all the time series. The metadata have all been used with the only exception of *distmod* and *target*.
- In the *CNN - joined_complete* architecture only the autoencoder is trained on all the time series, meanwhile the CNN has been trained only on the time series containing at least 10 points. The metadata have all been used with the only exception of *distmod* and *target*.
- In the *CNN - joined_reduced* architecture both autoencoder and CNN have been trained on the time series containing at least 10 points. The metadata have all been used with the only exception of *distmod* and *target*.
- In the *CNN - joined_reduced_redmeta* architecture both autoencoder and CNN have been trained on the time series containing at least 10 points. Moreover, the only metadata that have been used are *hostgal_specz*, *hostgal_photoz*, *hostgal_photo_err* and *mwebv*.
- In the *CNN - joined_splines* architecture splines, in place of the autoencoder, have been used to reconstruct the signal and then it has been trained on all the time series. The metadata have all been used with the only exception of *distmod* and *target*.
- In the *CNN - joined_splines_redmeta* architecture splines, in place of the autoencoder, have been used to reconstruct the signal and then it has been trained on all the time series. Moreover, the only metadata that have been used are *hostgal_specz*, *hostgal_photoz*, *hostgal_photo_err* and *mwebv*.
- In the *Exotic_splines* architecture, composed of stacked GRUs and Dense Layers, thanks to the preprocessing, done by using the splines in order to get uniformly distributed data, it has been possibile to train the architecture on the multivariate series of the 6 bands. The metadata have all been used with the only exception of *distmod* and *target*.

- The *Exotic_splines_redmeta* architecture is the same architecture of the *Exotic_splines* with the only difference that not all the metadata have been used, but just *hostgal_specz*, *hostgal_photoz*, *hostgal_photo_err* and *mwebv*.
  In order to find the best configuration for each model that would have reduced the loss we parameterized each model and then we ran a random search algorithm tuning over 150 different set of parameters. At the end, the saved hyperparameters, showed in Figure 4.1, correspond to the ones that produce the lowest loss for each model.



CNN - joined_complete+:

	f_filter	f_kernel	s_kernel	t_kernel	f_drop	s_filter	f_dense	s_drop	t_filter	test_loss	Epochs
0	96	3	1	4	0.3	156	96	0.1	128	0.989045	11

CNN - joined_complete:

	f_filter	f_kernel	s_kernel	t_kernel	f_drop	s_filter	f_dense	s_drop	t_filter	test_loss	Epochs
0	128	5	2	4	0.3	64	64	0.3	32	1.106	12

CNN - joined_reduced:

	f_filter	f_kernel	s_kernel	t_kernel	f_drop	s_filter	f_dense	s_drop	t_filter	test_loss	Epochs
0	128	3	2	5	0.5	256	96	0.3	64	0.978192	15

CNN - joined_reduced_redmeta:

	f_filter	f_kernel	s_kernel	t_kernel	f_drop	s_filter	f_dense	s_drop	t_filter	test_loss	Epochs
0	156	1	2	5	0.2	156	96	0.4	64	1.073	13

CNN - joined_splines:

	f_filter	f_kernel	s_kernel	t_kernel	f_drop	s_filter	f_dense	s_drop	t_filter	test_loss	Epochs
0	128	2	2	1	0.4	96	128	0.5	256	1.40086	17

CNN - joined_splines_redmeta:

	f_filter	f_kernel	s_kernel	t_kernel	f_drop	s_filter	f_dense	s_drop	t_filter	test_loss	Epochs
0	96	1	5	1	0.5	96	128	0.5	256	1.42834	23

Exotic_splines:

	f_gru	f_drop	s_gru	f_dense	s_dense	s_drop	t_dense	test_loss	Epochs
0	96	0.3	156	32	32	0.1	64	1.24274	15

Exotic_splines_redmeta:

	f_gru	f_drop	s_gru	f_dense	s_dense	s_drop	t_dense	test_loss	Epochs
0	96	0.3	156	64	64	0.1	128	1.30467	16

Figure 4.1: Best results of the various models

From the figure 4.1 we can conclude that by not using all the metadata we get a loss that is greater compared to the other models, which implies a significant importance of the information held by the metadata. Moreover, it is possible to notice that by preprocessing the data using the splines, which gives us an homogeneous sampling, it actually comes out that our neural network does not work as well as it does when raw data are used. Although, it is necessary to find a suitable architecture.

The architectures that present the lower losses, respectively of 0.99 and 0.97, are:

- *CNN - joined_complete+*
- *CNN - joined_reduced*

One would assume that the choice will fall on the *CNN - joined_reduced*, but this model has a drawback, because it was trained and tested not on the total of the objects but only on those containing more than 10 points in each band, which implies that when it will have to classify an object that does not reflect the training assumptions it might misclassify it. This model was created just to check how the architecture would work on cleaner data, so the loss that was obtained does not reflect the truth.

Hence, the architecture that we propose is the *CNN - joined_complete+*, that takes in account all the objects and all the metadata, without preprocessing them.

With a loss of 0.99 our position in the PLAsTiCC Challenge Chart is 123 on 1090.

Thanks for reading, Fiore Maffia, Christian Riccio, Antonio Elia Pascarella, Claudio Ciano