University of Beira Interior

Department of Computer Science

FACULDADE ENGENHARIA

Departamento de Informática

Analysis of Deep Learning-Based Frameworks For Searching Exoplanets

Author:

Diogo Pinheiro

Advisor:

Prof. Hugo Proença, PhD

June 14, 2020

Acknowledgments

I would like to express my deepest gratitude to my friends and family for having the patience to hear me rambling about the project for months and, most importantly, for the love and support.

To Prof.Hugo Proença for accepting my project proposal and for the constant support and guidance throughout the semester.

In addition, a word of appreciation to Prof.Paulo Moniz for the feedback about the proposed idea and for the extremely useful resources provided.

Contents

Contents **iii**

List of Figures **v**

List of Tables **vii**

1 Introduction **1**
- 1.1 Document Organization 2

2 State-Of-The-Art **3**
- 2.1 Exoplanet Detection 3
- 2.2 Literature Review 5
  - 2.2.1 Artificial Intelligence on the Final Frontier: Using Machine Learning to Find New Earths 5
  - 2.2.2 Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90 6
- 2.3 Machine Learning Algorithms 7
  - 2.3.1 K-Nearest Neighbors 7
  - 2.3.2 Logistic Regression 8
  - 2.3.3 Support Vector Machine 8
  - 2.3.4 Fully Connected Neural Networks 9
- 2.4 Deep Learning 9
  - 2.4.1 Convolution Neural Networks 9
  - 2.4.2 Long Short-Term Memory Neural Networks 11

3 Results and Discussion **13**
- 3.1 Data 13
  - 3.1.1 Creating the Training Set 14
  - 3.1.2 *Flexible Image Transport System* (FITS) 16
  - 3.1.3 *Astropy* 16
  - 3.1.4 Preprocessing 17
    - 3.1.4.1 Missing Values 19

3.1.4.2 Approach 2 20
3.2 Experiments and Results 21
3.2.1 Part 1 - Feature Extraction 21
3.2.1.1 Implementation 23
3.2.1.2 Results and Discussion 23
3.2.2 Part 2 - Deep Learning 25
3.2.2.1 Convolutional Neural Networks 26
3.2.2.2 Long Short-Term Memory Neural Networks . 27
3.2.2.3 Best Models 27
3.2.2.4 Evaluation Metrics 29
3.2.2.5 Results and Discussion 30
3.2.3 Part 3 - Interpretable Machine Learning 33
3.2.3.1 LIME For Time 34
3.2.3.2 Implementation 34
3.2.3.3 Results and Discussion 36

4 Conclusion And Future Work 39
4.1 Main Conclusions 39
4.2 Future Work 40

A Appendix A 41

Bibliography 45

List of Figures

2.1 Correlation between the planet transiting a star and the measured brightness [21]. 4
2.2 Diagram representing the processes done by both analyzed papers. 8
2.3 Illustration of a one dimensional CNN [24]. 10
2.4 Illustration on how zero padding allows the CNN to convolve the edges of the input vector. 11
2.5 Illustration of a LSTM gate [17]. 12

3.1 Representation of a light curve containing an exoplanet with Approach 1 (left) and Approach 2 (right). 13
3.2 Comparison between three noise removal algorithms. 14
3.3 Diagram that illustrates the data download process. 15
3.4 Structure of a FITS file. Adapted from [10]. 17
3.5 Representation of two views obtained from the same light curve. In both figures the graph shows the normalized flux (y-axis) in relation to of phase (x-axis). 19
3.6 Representation of thresholds in a light curve with two confirmed planets. PDCSAP FLUX values are defined in relation to time, with the total mean represented in black while the first and second standard deviations are green and red, respectively. 22
3.7 Best MLP Model Architecture. 28
3.8 Best CNN Model Architecture. 28
3.9 Best Dual CNN Model Architecture. 29
3.10 Best LSTM Model Architecture. 29
3.11 Representation of slices deactivation, using total mean, for a toy example in LIME For Time. 35
3.12 Explanation of two global view light curves. 37

A.1 Set of PC light curves with scaled PDCSAP FLUX in relation to time. 41
A.2 Set of NTP light curves with scaled PDCSAP FLUX in relation to time. 42
A.3 Set of AFP light curves with scaled PDCSAP FLUX in relation to time. 43

List of Tables

3.1 Comparison between reference and obtained values. 24
3.2 Reference values for the Deep Learning models. 31
3.3 Resulting values from the *Machine Learning* (ML) classifiers and Deep Learning models using Approach 1. 31
3.4 Resulting values from the Deep Learning models using Approach 2. 32
3.5 Performance of the dual *Convolution Neural Networks* (CNN) model using Approach 1. 32

Acronyms

AFP	*Astrophysical False Positive*
ASCII	*American Standard Code for Information Interchange*
AUC	*Area Under The Curve*
BJD	*Barycentric Julian Day*
CNN	*Convolution Neural Networks*
FITS	*Flexible Image Transport System*
HDU	*Header/Data Units*
KNN	*K-Nearest Neighbors*
LIME	*Local Interpretable Model-Agnostic Explanations*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
MLP	*Feed Forward Neural Networks*
NASA	*National Aeronautics and Space Administration*
NTP	*Non-Transiting Phenomenon*
PC	*Planet Candidate*
PDCSAP FLUX	*Pre-Search Data Conditioning SAP Fluctuation*
RNN	*Recurrent Neural Network*
RNN	*Recurrent Neural Network*
ROC	*Receiver Operating Characteristics*
SGA	*Stochastic Gradient Ascent*
SVM	*Support Vector Machine*
TCE	*Threshold Crossing Events*

Chapter 1

Introduction

Between 2009 and 2013, the Kepler space telescope observed around 200k stars using photometry. The main goal for this mission was to statistically determine the occurrence rate of planets that are similar to Earth - that is, planets that may contain carbon-based life just like ours. However, the data obtained by Kepler also allowed for the discovery of several compelling planetary systems [27].

No matter how precise the telescope observations are, the truth is that manually identifying exoplanets (using the provided data), becomes a significantly difficult task. Due to the extreme distances and some possible calibration problems, the obtained data does not always clearly show the existence of those planets. For those reasons, machine learning seems a great fit for the problem, as it will provide more precise and efficient results.

The main objective for this project is classifying lights curves, obtained by the *National Aeronautics and Space Administration* (NASA)'s Kepler space telescope, in order to identify exoplanets. Due to the significant amount of raw data and the difficulty in manually analysing that comes with it, machine learning classifiers and deep learning models will be used and compared.

The expected final result for this project is a computational prototype that is capable of identifying exoplanets through the analysed light curves. The implementation stage of this project will present some comparisons between neural network models and other classifiers, different preprocessing approaches and types of feature extraction. It will end with an explanation for the neural network model predictions.

1.1 Document Organization

1. The first chapter – **Introduction** – provides a context for this project, which includes the motivation and the objectives. It also presents the organization of the document.

2. The second chapter – **State-Of-The-Art** – contains some background information needed for this project, such as the methods used for exoplanet detection and a review of two papers that were used as reference. Additionally, it provides some information on the machine learning classifiers used in the third chapter.

3. The third chapter – **Results and Discussion** – describes the development process of this project, as well as the results and a comparison with the values described in both papers analyzed on the second chapter.

4. The fourth chapter – **Conclusions and Future Work** – presents the main conclusions for this project and some possible future work.

Chapter 2

State-Of-The-Art

This chapter provides a review of two papers that were fundamental for the development of this project, as well as a short explanation on exoplanets and some of the methods used to identify them. It also shows some of the theory behind the *Machine Learning* (ML) and deep learning architectures used in this project.

2.1 Exoplanet Detection

In general, exoplanets are planets that orbit a star that is not our sun, that is, planets that are located outside of our solar system. There is an immense variety of planets, especially in terms of their orbit, size, atmosphere and their composition [2].

Between 2009 and 2018, the Kepler space telescope's observations allowed for the discovery of several exoplanets. During this period, more than half a million stars were observed and about 2.700 were identified [5].

The main interest in the study of these planets is the search for life similar to the one we know. For that, NASA and other space agencies are looking for Earth sized planets that orbit a star similar to the sun in a possibly habitable zone known as the Goldilocks Zone [13].

There are several approaches for the search of exoplanets. Direct observations are not always possible, and as a result astronomers found a clever way to identify these planets, monitor a star and look for the possible star-planet interactions. Three observation methods stand out from the rest: radial velocity, gravitational microlensing and transit photometry.

The first approach is one of the most effective, it looks for small changes in the observed star's position. This method relies on the fact that stars are affected by the gravitational tug of an orbiting planet. These movements can result in a change of the star's normal light spectrum and this is what astronomers are looking for [11].

The gravitational microlensing method is, so far, the best method for identifying exoplanets at extremely long distances. This method relies on a principle originally introduced in Einstein's General Theory of Relativity, a star appearing directly behind another one (in the observer's perspective) will create an effect known as the Einstein's disk, in which the emitted light rays are amplified and create what looks like a disk. This effect last as long as the stars remain aligned [9].

At last, the transit method relies on the brightness levels variation to identify exoplanets [15]. A practical example of this would be observing a lamp when a mosquito suddenly passes between the observer and the lamp, creating a shadow or at least a reduction in the brightness levels. This is the method that will be used for the project.

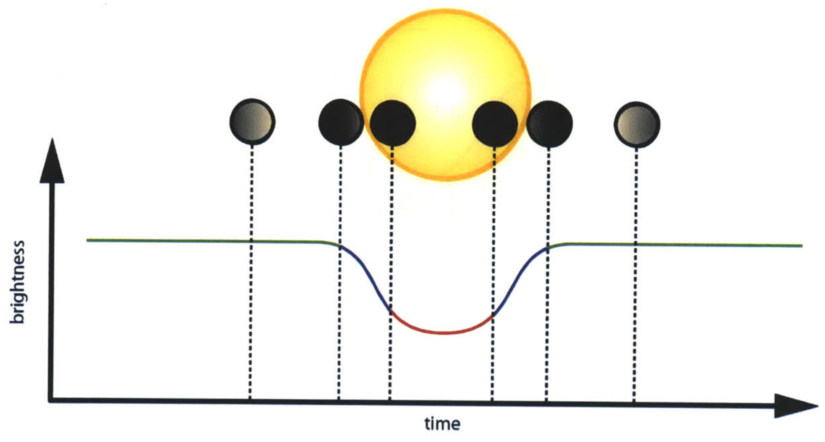


Figure 2.1: Correlation between the planet transiting a star and the measured brightness [21].

2.2 Literature Review

This section provides an analysis of the two papers that were fundamental for the development of this project. This analysis will be mostly focused on the preprocessing and ML algorithms implementation stages. The results of both papers will later be compared to the ones obtained in this project (Section 3.2.2.5).

2.2.1 Artificial Intelligence on the Final Frontier: Using Machine Learning to Find New Earths

To eliminate noise and normalize the obtained light curves, a few data preprocessing steps were needed. The presence of low frequency oscillations in the light curves may lead to some analysis errors because, in order to get the best possible result, the peaks need to be as clear and accentuated as possible. To solve this problem, a moving average filter was applied as basis for the following percentage-change calculation of each point. The moving average was considered using the 15 nearest points on both sides, which represents a total of about eight hours. The final result converts the data from electrons-per-second (*Pre-Search Data Conditioning SAP Fluctuation* (PDCSAP FLUX)) over time to percentage-change over time, which as a result eliminates irrelevant fluctuations.

Having normalized curves, a set of thresholds for peak identification were defined. Two different thresholds were set, one and two standard deviations below the mean percentage-change of the given data. The following process was to iterate over each data point, in order to identify and classify the three expected peak types: strong (likely to correspond to exoplanets), medium (may correspond to an exoplanet) and weak (high probability of being just noise).

This categorization was possible by analysing in what interval (between the mean and thresholds) the value is positioned, for example, it is considered as a strong peak if its value is less than the second standard deviation below the mean.

Because of the difficulties in properly aligning time-series data to extract features, it can be useful to just remove the time factor. The main goal here is to determine the differences between the more accentuated peaks and the other variations. Additionally, it is also necessary to obtain other relevant information, such as:

- Periodic consistency: exoplanets transits should be quite periodic because their orbital rate when orbiting a star is rather constant.
- Short peaks: the transit time should be relatively short because both the planet and Kepler are in constant movement.

To obtain the necessary attributes to properly identify exoplanets, 325 features in total were extracted using peak identification and some other signal analyzes techniques. All of them based on the calculation of the mean and standard deviation values used with the previously mentioned thresholds.

Knowing that this is a supervised learning problem and having already obtained the features and corresponding classes of each light curve, the logical next step was to to classify the data using some ML classifiers: *K-Nearest Neighbors* (KNN), logistic regression, softmax regression and *Support Vector Machine* (SVM).

The approach taken was to manually implement this algorithms, using different optimization methods for each classifier (e.g *Stochastic Gradient Ascent* (SGA) was used in logistic regression).

2.2.2 Identifying Exoplanets with Deep Learning: A Five-planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90

The main goal of this project was to identify exoplanets using deep learning neural networks. The data used was obtained from a set of *Threshold Crossing Events* (TCE)s and a table which contains a classification label for each kepID (Kepler's target star ID) that is associated to a folder with light curves files. There are three main categories: *Planet Candidate* (PC), *Astrophysical False Positive* (AFP) and *Non-Transiting Phenomenon* (NTP). In total, 3600 PCs, 9596 AFPs and 2541 NTPs were analyzed.

However, this is a binary classification problem, so these three categories had to be reduced to only two: planet (PC) or not-planet (AFP and NTP). After obtaining the light curves corresponding to each row of the table, some preprocessing steps to the data were necessary in order for it to be used as input for the neural network.

For each TCE in the training set, all points corresponding to other confirmed planet transits in the system were removed. This was followed by the light curve normalization using the spline interpolation to remove low frequency oscillations. This was done by adjusting a spline as basis to the curve and divide it by the best adjusting spline. The signal is then subjected to folding and binning, two signal transformation processes that will create the 1D vector input.

Three architectures of neural networks were considered: linear (no hidden layers), fully connected and convolutional. For each one, three types of data were considered: global view, local view and both. The hidden layers used a ReLU (linear rectification) activation function and a sigmoid activation function for the output layer.

In the implementation stage, the training dataset was augmented through the application of random horizontal reflections to the signal. Additionally, it was also applied a dropout layer to the fully connected networks, which prevents the model from becoming too dependent of any feature and consists on randomly choosing some output neurons and then dropping them for each layer during training.

To automatically tune the hyper-parameters, a Google-Vizier system was used. After the selection of this parameters, ten independent copies were trained with different random initialization parameters. The average of this outputs was used to analyze the models, this improves the performance because different versions of the same configuration can perform better specially when the dataset is relatively small.

2.3 Machine Learning Algorithms

Machine learning allows systems to identify patterns in data and consequently learn and make decisions from it with just the slightest human intervention [14]. There are several types of learning, but they can be summarized into three major categories: supervised, unsupervised and reinforcement learning. For this project, only supervised algorithms were used - that is, algorithms that depends on labeled input data to learn a function that produces a fitting output [7].

2.3.1 K-Nearest Neighbors

This is probably one of the simplest algorithms and it can be used for both classification and regression problems. This algorithm assumes that elements

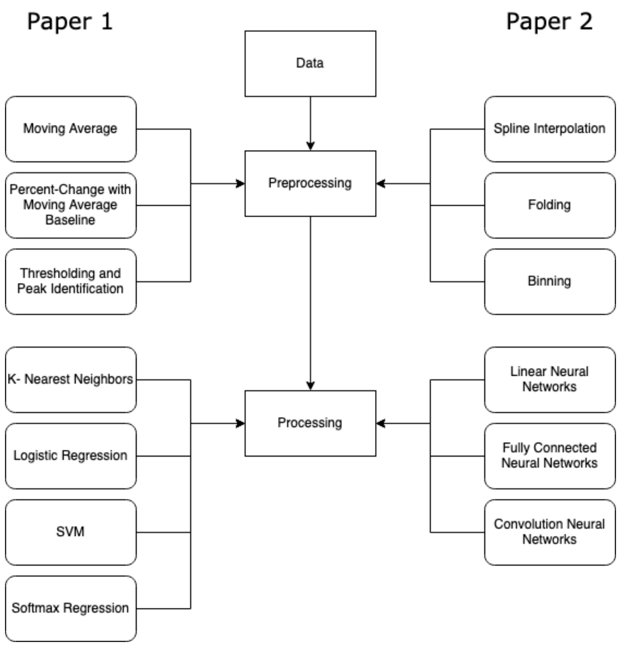


Figure 2.2: Diagram representing the processes done by both analyzed papers.

of a class are close to each other in the feature space and, for that reason, the main goal is to determine the class based upon the distances between all data points and the target point. After sorting the collection of distances, it will select the first k closest points -the neighbors- and get their labels. The class that is most represented in those neighbors is the one that will be associated with the target point [7].

2.3.2 Logistic Regression

This method uses the logistic function (also called sigmoid function) that uses an equation as representation. It combines weights with the input values to predict the output results and it can only be used when the target is categorical. The data is fitted into a linear regression model, which then can be determined by a logistic function [6]. To predict which category a class belongs to, a threshold is usually set, creating a sort of separation that will determine the class based upon the result value.

2.3.3 Support Vector Machine

SVM is another algorithm that can be used for both classification and regression, its main goal is to define a N-dimensional hyperplane that successfully separates classes and is able to classify the data [12]. The problem is that there are many possible hyperplanes that can accomplish this task, for that reason, this algorithm finds a plane that has the maximum possible margin between data points representing both classes.

Support vectors are defined as the data points closer to the hyperplane margins and they are able to influence its orientation [12]. To classify the data, the algorithm only needs to verify the point's position in relation to the hyperplane.

2.3.4 Fully Connected Neural Networks

Neural networks consist of neurons that are grouped in layers and for which the outputs (called activations) of one layer are connected directly to the input of the next, possibly having hidden layers between them. This hierarchical arrangement ends in the final layer where the outputs are the predictions made by the network.

There are different activation functions, all of them nonlinear, such as the sigmoid function, hyperbolic tangent and linear retification ([27]). If the problem is binary, then it is common to use the sigmoid function with normalization (change the values so that they are in the range [0,1]).

2.4 Deep Learning

Deep learning is considered to be a sub-field of ML algorithms that uses multiple layers to acquire features from input data [18]. This data usually does not need to be processed and the features obtained from it cannot be understood by humans. As described in [29] by Yann LeCun, Yoshua Bengio and Geoffrey Hinton, the most important attribute about deep learning is that the features that it finds are not designed by humans, the networks learn from the data using an all-purpose procedure. This section provides a theoretical overview of the deep learning architectures used in the development stage of this project.

2.4.1 Convolution Neural Networks

Convolution Neural Networks (CNN) are able to create multiple copies of the same neuron to model substantial amounts of data by using weight tying techniques [22]. The number of parameters that the model needs to use in order to successfully learn is relatively small (specially when compared with the number of parameters required by the fully connected neural network). This happens because CNNs are able to learn the local features from the input and then makes use of spatial structure, which means that each feature will only need to be learned once [27].

Usually a CNN consists of convolutional and pooling layers, the neurons are aggregated in the former and the outputs are presented in the latter (Figure 2.3). This project analyzes time-series data and, as such, it can only be processed by an one dimensional CNN. The operation that takes the input vector to the output vectors is called convolution, but is also known as discrete cross-correlation [27].

$$s(t) = \int x(t)w(t-\tau)dt = (x * w)(t) \tag{2.1}$$

The convolution theorem is typically denoted with an asterisk and the x(t) value of the equation represents the input while the w(t) corresponds to the kernel and the output, s(t), is the feature map.

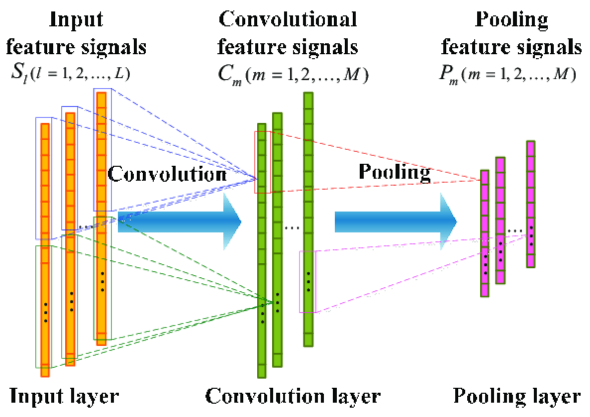

Input feature signals $S_l (l = 1,2,\ldots,L)$
Convolutional feature signals $C_m (m = 1,2,\ldots,M)$
Pooling feature signals $P_m (m = 1,2,\ldots,M)$
Convolution
Pooling
Input layer
Convolution layer
Pooling layer

Figure 2.3: Illustration of a one dimensional CNN [24].

The pooling layer summarizes the features in chunks of map features by taking the average (average pooling) and maximum (max pooling) values of each closest neighbors and grouping these values. This is helpful if the presence of a feature is more important than where that feature is, which prevents the network from resulting in different feature maps when the input values are modified [27].

There are different types of pooling operations but all of them apply a filter that is smaller than the feature map, so the output is always smaller than the input, each feature will be reduced by a factor defined by the operation. This ability is called invariance to local translation [1].

A CNN filter is not able to convolve the edges, for that reason, a given input produces an output with smaller dimensions. This is a problem because this edges can contain valuable information and, additionally, convolving with more filters makes the output size become increasingly smaller and basically meaningless [19]. The size of the output (o) can be calculated as 2.2 for the one dimensional CNN and 2.3 for two dimensions. In these equations, v represents the input vector size and f the filter size.

$$o = v - f + 1 \tag{2.2}$$

$$o = (v - f - 1) * (v - f + 1) \tag{2.3}$$

The solution for this problem is called zero padding, to properly convolve the edges of the given vector it adds zero valued elements to the borders (Figure 2.4) allowing the CNN to preserve the input size.

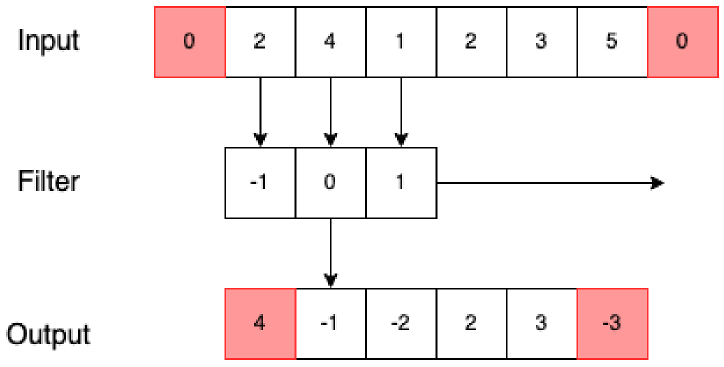


Figure 2.4: Illustration on how zero padding allows the CNN to convolve the edges of the input vector.

2.4.2 Long Short-Term Memory Neural Networks

A *Long Short-Term Memory* (LSTM) neural network is a modified version of *Recurrent Neural Network* (RNN)s that can be looked as some sort of feed forward neural network but with internal memory [17]. RNNs are implemented in a way that allows a copy of the output to be sent back to into the network's input. As a result, all inputs are, in a way, interconnected [17]. One of the biggest disadvantages of this network is that it can not handle long input sequences due to having limited memory because of vanishing gradient problems [22].

LSTMs, however, were explicitly designed to avoid these problems. These networks use back-propagation and the long-term memory dependency is ensured by having a cell state that carries information throughout all cells [22], making it easier to remember past data. This kind of network is usually constituted by three gates: input gate, forget gate (decides what details should be discarded by the block) and the output gate [17].

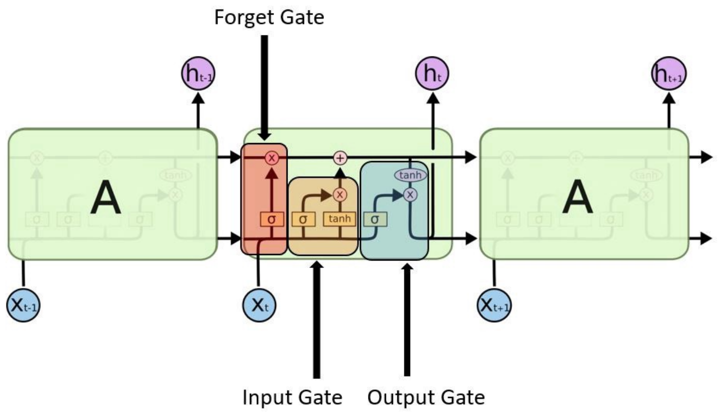


Figure 2.5: Illustration of a LSTM gate [17].

As seen in Figure 2.4.2, the repeating module of a LSTM contains four interacting layers (represented by the yellow box) and five pointwise operations (pink circles). Additionally, in this figure it can also be seen the cell state which is the horizontal line running through the module [16].

Chapter

3

Results and Discussion

This chapter describes the development process of this project, as well as the results and a comparison with those obtained in both papers analyzed on the second chapter.

3.1 Data

There are a number of ways one can preprocess the input data and, as such, in this project the approach taken by [27] (Approach 1) will be compared to other involving an additional step (Approach 2).

As seen in Figure 3.1, the signal obtained using Approach 1 still has a substantial amount of noise. With Approach 2 the main goal is to feed a cleaner signal to the classifiers.

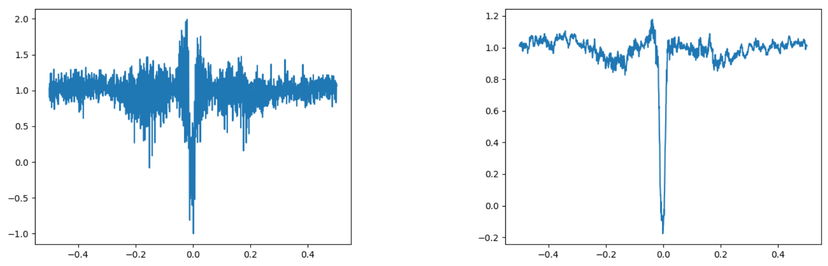


Figure 3.1: Representation of a light curve containing an exoplanet with Approach 1 (left) and Approach 2 (right).

At first, three major candidates for Approach 2 were chosen to remove the unwanted high frequency oscillations in the signal: moving average filter, time variation filter and Savitsky-Golay filter. For a proper comparison, these three types of digital filters were applied to the same light curve (3.2). All three of them successfully filtered out the noise but, due to the similarities, it was decided that only one of them would be used for this project. In the figure it can be seen that the Sovitsky-Golay filter presents a slight case of Gibbs phenomena (a small ripple before the more significant value drop) probably due to using a Fourier transform, and the time variation filtered signal is slightly more squared than the other two so, for that reason, the moving average filter was chosen for Approach 2.

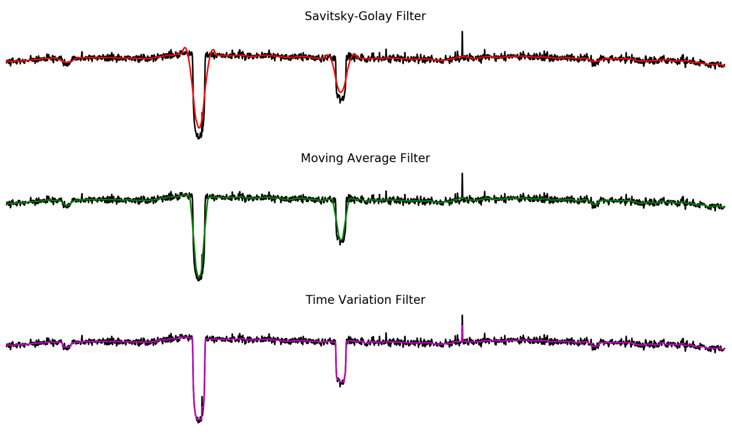


Figure 3.2: Comparison between three noise removal algorithms.

3.1.1 Creating the Training Set

Similarly to having two approaches in the preprocessing stage, there are two possible ways to create a training set. In the first one, code from [27] and [22] was adapted to download and create the data tables, while the other one used an already existing *Python* library. The latter uses the same processes as the former, but with the simplification of having it all already implemented and ready to use. In this sub-section, we will go through the steps taken to download data from the NASA repository and then create a table using the second approach. The resulting datasets will then be compared in section 3.1.4.

The TCE labelled data was obtained from the Autovetter Planet Candidate Catalog[1], corresponding to quarters 1 to 17 in DR24. This data contained seven features: rowID, kepID (ID of the target star), tce_plnt_num (TCE number of the star), tce_period (period of the detect event, in days), tc3_time0bk (time corresponding to the first *Barycentric Julian Day* (BJD) event), tce_duration (duration of the event, in hours) and av_training_set (labels).

In order to obtain the long cadence light curves associated with all kepids in the TCE table, a script was created. This script uses the *wget* command to download the kepid (Kepler's target star ID) corresponding light curve files from the Mikulski Archive for Space Telescopes[2].

This script (Figure 3.3) iterates over each row, downloads the corresponding kepid files and organizes them in a folder with its first four digits. Inside this folder there is another with the total kepid number, which contains the light curves in *Flexible Image Transport System* (FITS) file format (see 3.1.2 for more information regarding the format). The light curves are separated in four quarters but not all files contain all the quarters, which means that not all folders have the same number of files.

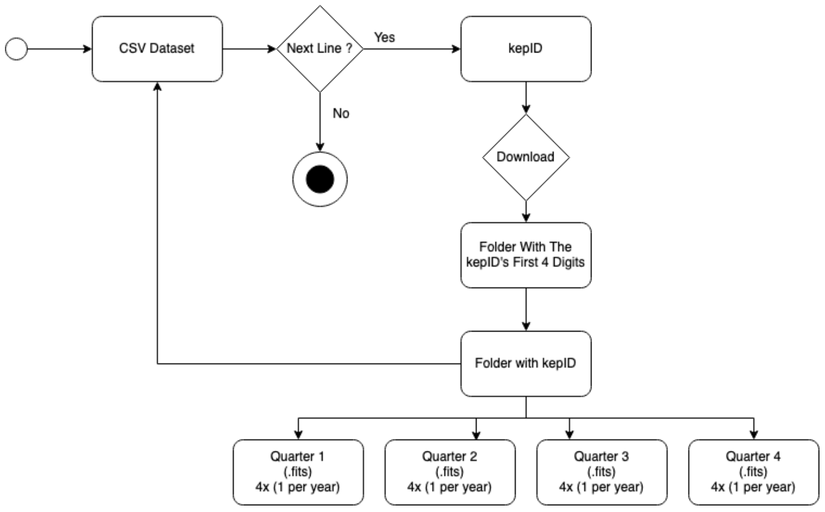

CSV Dataset
Next Line ?
Yes
No
kepID
Download
Folder With The kepID's First 4 Digits
Folder with kepID
Quarter 1 (.fits) 4x (1 per year)
Quarter 2 (.fits) 4x (1 per year)
Quarter 3 (.fits) 4x (1 per year)
Quarter 4 (.fits) 4x (1 per year)

Figure 3.3: Diagram that illustrates the data download process.

For each light curve, only the PDCSAP FLUX was used, this is the flux in units of electrons-per-second that the optimal aperture pixels obtained by the Ke-

[1]https://exoplanetarchive.ipac.caltech.edu
[2]https://archive.stsci.edu/

pler telescope contains. These values were all processed by NASA beforehand to remove some known error/noise sources and signals [20].

One of the major problems found in the light curve signals is that they are quite noisy (low-frequency drifting and high-frequency noise), which is a result of extreme distances and constant movement of the Kepler space telescope. As mentioned before, Approach 2 in 3.1.4 will apply a moving-average filter to remove these oscillations while trying to retain the most important features of the signal.

To read and manipulate data in FITS file format, the most common used library is *Astropy* (see 3.1.3).

3.1.2 FITS

FITS is a standard digital format used in astronomy, usually for manipulation and storage of data. This format is more than just an image file format, it contains photometric information on spatial calibration and image associated metadata [4].

This format was developed to store multidimensional arrays (images) and two dimensional tables (columns and rows). It basically comprises a group of segments called *Header/Data Units* (HDU), the first one named "Primary HDU" and the following HDU segments named "Extensions", which usually consist of three data types: images, *American Standard Code for Information Interchange* (ASCII) tables and binary tables [4].

Each HDU segment contains a header unit and, optionally, a data unit [10]. The structure for this file format is represented in Figure 3.4.

3.1.3 *Astropy*

Astropy is an open source library which makes the data handling process, used in astronomy and astrophysics, easier and provides some tools for reading and manipulating FITS format files.

To read FITS format files, *Astropy* contains the *open()* function that returns an HDUList, which is basically a list that collects HDU objects. Having this list, the package offers some data analyzes functionalities, for example the command *info()* shows a summary of the opened file content [3]. The access to this content is similar to the methods used in Python dictionaries.

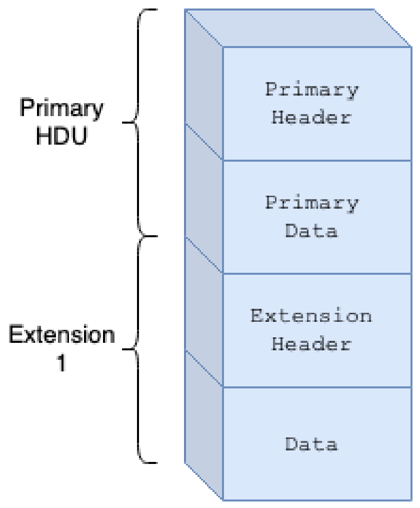


Figure 3.4: Structure of a FITS file. Adapted from [10].

```
>>> hdul.info()
Filename: ...test0.fits
No.    Name      Ver    Type       Cards   Dimensions
   Format
  0  PRIMARY       1 PrimaryHDU     138    ()
  1  SCI           1 ImageHDU        61    (40, 40)    int16
  2  SCI           2 ImageHDU        61    (40, 40)    int16
  3  SCI           3 ImageHDU        61    (40, 40)    int16
```

3.1.4 Preprocessing

The *LightKurve* library provides a tutorial [8] in which it is shown the different processes needed to successfully clean and process the data. The first step, once we have the data, is to clean it from possible missing values and outliers.

After that, it is time to solve one problem associated with this data, one signal may correspond to various types of events, meaning that one light curve can have, for example, more than one planet candidate. For this reason, a mask is applied to the signal. This can be done using the orbital period, transit duration and transit epoch obtained from the TCE table [8].

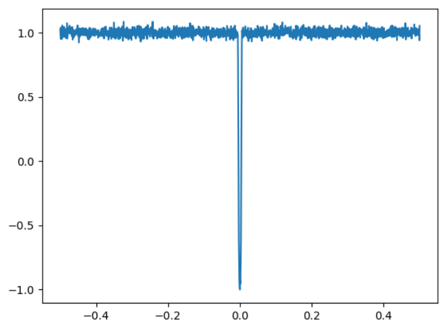
```
temp_fold = lc_clean.fold(period, t0=t0)
fractional_duration = (duration_hours / 24.0) / period
phase_mask = np.abs(temp_fold.phase) < (fractional_duration
    * 1.5)
transit_mask = np.in1d(lc_clean.time, temp_fold.
    time_original[phase_mask])
```

At this stage, the signal is ready to be flattened. This is done by applying a basis spline and dividing it by the spline that best fits the curve. Additionally, interpolation is also used to preserve the transits from the signal. After folding, a global and local view is generated. This views correspond to binning the signal to a fixed length, global has 2001 values and local has 201 values.

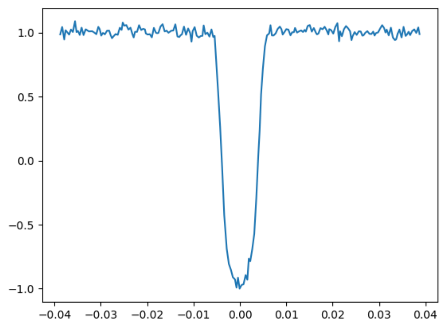
Data folding is a technique used in astrophysics and cosmology, which involves cutting and folding the data into multiple sub-series with a previously defined time period. The resulting data is plotted in terms of phase instead of time. The binning stage allows to group data into a static number of bins, this avoids the problem of different light curves having different sizes.

However, despite doing the exact same process as in [27], the final number of light curves present in the obtained dataset was not the same. The approach that used the *LightKurve* library resulted in 13773 rows while the first approach with 15542 was closer to the 15737 from the reference. This difference was probably caused by some limitations of the library for the second method and having slightly different TCE tables for the first method.

The dataset obtained from using an adaptation of the already existing code from [27] and [22] was the one chosen to train the data throughout the project because it allows for a more reliable comparison with the references, although this difference might be enough to in the end reach also different results. In total, the data contains 3599 PC and a combination of 11942 NTP and AFP light curves (a small set of each category can be seen in A).



(a) Global View (2001 values)



(b) Local View (201 values)

Figure 3.5: Representation of two views obtained from the same light curve. In both figures the graph shows the normalized flux (y-axis) in relation to of phase (x-axis).

3.1.4.1 Missing Values

One of the biggest obstacles in data analysis is the presence of NaN values, which represents the absence of a value. This values are extremely dangerous because they can single-handedly change some metrics and, therefore, present wrong information.

There are a number of ways to deal with this problem. However, some solutions are not ideal, for example removing a row that contains a NaN value will result in lost information. In this project, the chosen solution was to use sigma clipping which iterates over the data and masks the values that are above a certain standard deviation (the sigma).

3.1.4.2 Approach 2

The second approach uses the moving average as a way to remove the noise present in the signal. The moving average is calculated by finding the mean of the values contained in a certain window of size N followed by a shift forward in the subset. This method reduces the size of the data and, because of that, if we try to insert this values in the same initial time period it will originate some problems in the edges.

The implementation of this filter, for this project, uses convolution despite not being the most common way. The main difference of this approach to the conventional moving average is that the former uses an arbitrary linear combination (i.e. multiply each element by their coefficient) and sum the results.

To solve the edges problem, a *Numpy* padding functionality was used to, together with the valid mode in the convolution function, produce a more acceptable result.

```
data_padded = np.pad(data, (N//2, N-1-N//2), mode='edge')

return np.convolve(data_padded, np.ones((N,))/N, mode='valid
    ')
```

After smoothing the signal, the light curves may have different scales so the values need to be normalized. For this, the *normalize()* function in the *Scikit-Learn* library was used.

3.2 Experiments and Results

This section is divided in three major parts, providing some details regarding the implementation stage of this project. The main goal is to describe with some detail all the steps that followed the data preprocessing previously explained. The code is also available online[3].

3.2.1 Part 1 - Feature Extraction

The first part of this section is a somewhat special case, at least regarding the usage of the obtained data. Unlike the other two parts, the main goal for this one is to create our own training set. To do that we take the already preprocessed data and extract some of its the most important features. This is an adaptation of the steps described by Abraham Botros in [20].

As it was mentioned in Chapter 2, the transit method measures the brightness changes of a star. However, exoplanets are not the only space objects capable of producing such variations. A peak in the signal could also correspond to a binary star systems or, amongst others, pulsating stars [20].

There are usually two main characteristics that define an exoplanet, firstly, exoplanets orbit their respective star with a regular orbital rate, therefore the peak appearance in the signal should be periodically constant. Secondly, both Kepler and the exoplanet are moving so the peaks should be brief [20].

To extract features from the data, a set of three thresholds were created. These thresholds represent the total mean and the first and second standard deviation below this mean. For simplification purposes, three categories were created (Figure 3.6):

- Strong: Values smaller than two standard deviations below the mean;
- Medium: Values smaller than one standard deviation below the mean;
- Weak: Values smaller than the mean but above the first standard deviation.

For this dataset, a total of 16 features were obtained for each light curve:

[3]https://github.com/DiogorPinheiro/Identifying-Exoplanets-Using-ML

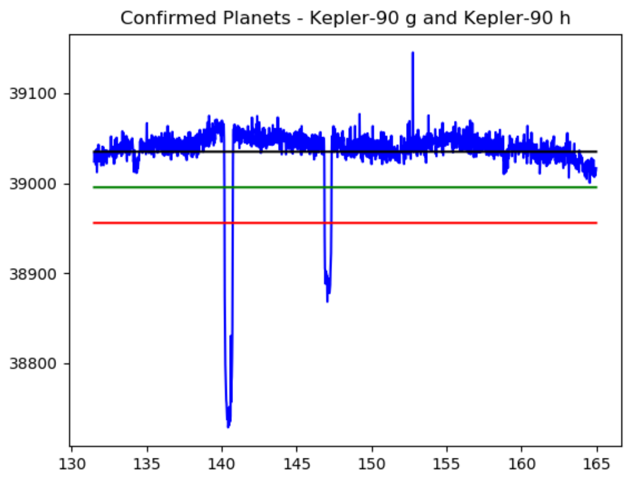


Figure 3.6: Representation of thresholds in a light curve with two confirmed planets. PDCSAP FLUX values are defined in relation to time, with the total mean represented in black while the first and second standard deviations are green and red, respectively.

1. Number of peaks in all three categories;
2. Global median, mean and standard deviation;
3. Mean and standard deviation for strong and medium peak widths;
4. Maximum valued peak (measured value and percentage in relation with the mean);
5. Mean and standard deviation for each category;
6. Mean and standard deviation for the weak peak widths;

Items three to seven were obtained for both standard deviation thresholds. To retain some additional possible relation between these features, a pairwise product was also calculated for all.

This part (and the whole project) is a binary classification problem so, for that reason, the four types of labels were reduced in half to planet(1) and not planet(0). The first category corresponds to the PC while NTP and AFP make the not planet category. The unknown light curves were ignored from the dataset because they would only bring uncertainty to the final result.

Finally, all dataset columns were normalized using the Scikit-Learn preprocessing *normalize* function. The final result is a csv file with 138 columns (including kepid and label) and 10598 rows.

3.2.1.1 Implementation

To evaluate how the created data describes if the light curve contains a exoplanet or not, three ML classification algorithms were used: KNN, logistic regression and SVM.

In order to find the best possible result with each algorithm, it is important to first apply some hyper-parameter tuning. For the KNN, an *Area Under The Curve* (AUC) was used to measure the best possible combination of parameters. This evaluation metric describes how capable the model is to distinguish between classes using a *Receiver Operating Characteristics* (ROC), which is a probability curve.

To tune the norm of penalization and the regularization strength for the SVM, a grid search algorithm was used. This procedure implements an exhaustive search for the best parameters and evaluates them with a fit and score method.

```
parameters = {'kernel': ('linear', 'rbf'), 'C': [1, 10]}
svr = svm.SVC(probability=False)
model = GridSearchCV(svr, parameters,n_jobs=4, verbose = 5)
```

To optimize the logistic regression results, a grid search algorithm was also used. For this type of classifier the parameters that needed to be tuned were the norm of penalization and the regularization strength, the last one implies that with a higher value the regularization becomes weaker.

```
grid = {"C": np.logspace(-3, 3, 7), "penalty": ["l1", "l2"]}
logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg, grid, cv=10)
```

3.2.1.2 Results and Discussion

Finally, after all these steps, the results were obtained by training the data and evaluating with the *Scikit-Learn score()* metric function which will internally create a set of predictions for the test data and then compare its accuracy to the true labels. The final value is the result of a simple calculation that will be explained further down this chapter in 3.2.2.4. For this part of the project, following the reference paper, only the accuracy value was obtained although this metric by itself is not enough to properly evaluate a model.

Table 3.1: Comparison between reference and obtained values.

Classifier	Results	Reference
SVM	0.776	0.833
KNN	0.762	0.839
LReg	0.777	0.845

As seen in the above table, the final results were lower than the reference values, but the difference was not quite as significant as expected in the beginning because the feature extraction process done in this part was simply an adaptation of the one described in the paper and not all features were implemented.

Additionally, it is also likely that a difference in the training data may be related to this discrepancy of values. In [20] about 2000 light curves were used while this project used 10k more than that.

3.2.2 Part 2 - Deep Learning

The second part of this project has as its main goal the comparison between training data in Deep Learning models and other more "common" machine learning classification algorithms, as well as between the two preprocessing approaches described in 3.1.4. The following pages will describe how the Deep Learning models architectures came about and, in the end, the obtained results will also be compared to a reference paper ([27]) and a master's dissertation ([22]).

To find the best possible model and hyperparameter combination, a manual trial and error approach would not be efficient because it would be time-consuming and the probability of finding the optimal results would be somewhat low, thus a more automatic model building approach was needed. There are several options in the market, but since the models were built using the *Keras* API (with a *TensorFlow* backend) the one chosen for this project was *Keras-Tuner*[4], one of its libraries, which performs hyperparameter tuning using *TensorFlow 2.0*.

Keras-Tuner performs a loop to find best parameter combination, which includes setting a set of parameter values and then train them and evaluate the model, after iterating over a certain number of cycles it compares the evaluation results and outputs the best one. This library offers three tuning methods: random search, Bayesian optimization and hyperband. The latter was used to build the models described in the following sub-sections, this method speeds up the tuning process by using early stopping and creates a specific schedule of iteration per configuration, running some random configurations within this time period, taking the best performers and then running them for longer periods of time.

Implementing this library was quite easy, a *HyperModel* class object was created and defined as shown in the code below. This class has as its inputs the model, the number of output classes and the input shape, which is usually a 3 dimensional parameter set defined as (number of samples, number of timesteps, number of features). Inside the class there are three main methods: init, build and conf. The first one initializes the input variables, the second builds and evaluates the best parameter combination and the last one returns the input configuration to allow the model to be used later. Once this class is called, it returns the best model configuration (which will then be saved as a hdf5 file), as well as those input configurations mentioned before.

[4]https://keras-team.github.io/keras-tuner/

```
class CNNHyperModel(HyperModel):
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes

    def build(self, hp):
        model = keras.Sequential()
        model.add(
            tf.keras.layers.Conv1D(
                filters=hp.Choice("filter1", [8, 16, 32, 64,
                    128, 254], default=64),
                kernel_size=hp.Choice('kernel1',  [3, 5, 7,
                   9], default=5),
                activation="relu",
                input_shape=self.input_shape,
            )
        )
        ...
        return model

    def get_config(self):
        return {
            'input_shape': self.input_shape,
            'num_classes': self.num_classes
        }
```

The code above is a snippet of the CNN model builder, it displays how the *HyperModel* class is structured and how one can easily implement a set of parameters for the hyperparameter tuning.

3.2.2.1 Convolutional Neural Networks

Studying known and proved CNN models like *AlexNet* and *VGG* provided some insight on what usually follows a convolutional layer. The most common layers inserted after the output of a CNN layer are max pooling and dropout.

Max pooling provides a way to reduce the data dimensionality, which also reduces computational costs. This layer (just like all layers implemented in this project) already exists in *Keras* but the same result could be obtained by just increasing the stride parameter of the CNN layer, although this would increase the computing time and pooling still has other inherent advantages like translation invariance.

The dropout layer is one of many regularization techniques available, this is a simple but effective way to prevent the network from overfitting, that is the model wrongly giving too much weight to some data features resulting in a significantly worse performance when presented with new unseen data.

3.2.2.2 Long Short-Term Memory Neural Networks

According to [28] using both dropout and batch normalization often leads to unexpected final predictions unless some conditioning is applied to avoid variance shifts (inconsistency in statistical variance between train and test mode) which leads to incorrect predictions. Two possible solutions are applying the dropout after the last normalization block, which avoids scaling on the filter outputs, and to modify the dropout function so that it becomes less sensible to variance. However, in this model only dropout was used for data regularization.

It is also worth to mention that the dropout layer might also bring some unwanted problems, in order to avoid becoming too dependent on some neurons this regularization method adds noise to them. This sometimes causes long-term memory loss in LSTM networks. One possible solution would use recurrent dropout which masks the connections in recurrent units instead of masking the inputs like the regular dropout [23]. Nevertheless, this regular dropout method was applied to the model and didn't affect the final result.

Additionally, although often overlooked initialization bias can significantly contribute to better results in neural networks. For LSTM networks, a positive initialization bias was found to be the ideal value [26]. In *Keras* this is achieved by setting the unit_forget_bias parameter to True, which should also be accompanied by setting the bias_initializer to 'zeros'.

In terms of activation, although it can be considered as non-necessary because a typical LSTM cell already contains a few non-linearities provided by the tanh and the three sigmoid functions, better results were obtained when using a LSTM block followed by PReLU.

3.2.2.3 Best Models

As mentioned in 3.2.2, the best combination of parameters and number of blocks were obtained using *Keras-Tuner*. In this sub-section it is shown, and briefly explained, a visual representation of the best models found for the *Feed Forward Neural Networks* (MLP), LSTM and CNN architectures. All represented models have a global view input, except for the dual CNN model which receives both global and local views.

Starting with the most simple neural network, the best combination found for the MLP (Figure 3.7) consists of three blocks of a Dense layer with ReLU activation followed by the output, a Dense layer with sigmoid activation.

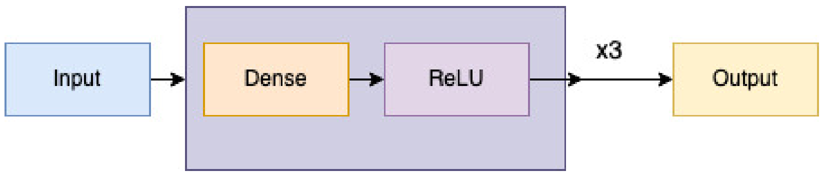


Figure 3.7: Best MLP Model Architecture.

When building a *Keras* model, there are two possible ways to do it: functional or sequential. The latter has a layer by layer approach, not allowing the model to have more than one input or shared layers, while the former allows for a more flexible approach to build models. As mentioned in the beginning of this sub-section, two types of CNN models were built, a sequential network and a functional dual input network.

The best sequential CNN model (Figure 3.8) has three convolutional blocks (the first one being the input layer), which includes a 1D CNN cell followed by batch normalization, max pooling, dropout and PReLU activation layers. Then the signal is flattened and it goes through two regular densely-connected neural networks with a dropout and PReLU activation layers between them, with the last one being the output layer with sigmoid activation.

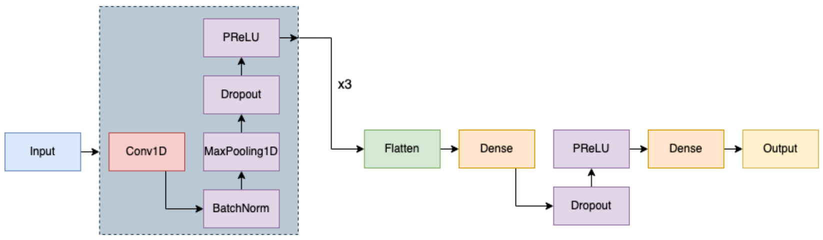


Figure 3.8: Best CNN Model Architecture.

For the functional model (Figure 3.9) , we have two different input thus the configuration itself will consist of a concatenation of two CNN pathways followed by three fully connected network blocks (dense, batch normalization and dropout) and the output. The global view input has almost three times more convolutional block layers than the local view input, but both blocks consist of the same type of layers: CNN, max pooling and dropout.

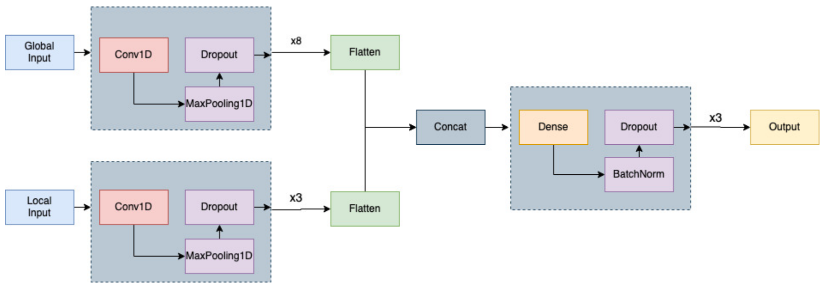


Figure 3.9: Best Dual CNN Model Architecture.

Finally, the best LSTM model (Figure 3.10) has the same overall structure of the sequential CNN, it includes two main types of blocks: the first one occurs four times and is defined by the LSTM cells followed with a PReLU activation and a dropout layer, then the signal is flattened and goes through four blocks of dense layers followed also by a PReLU and dropout layer. The output layer a activation of type sigmoid, which is the best fitted for binary cross-entropy results.

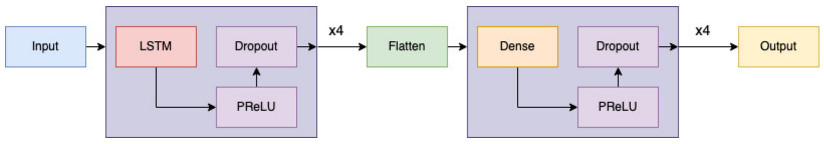


Figure 3.10: Best LSTM Model Architecture.

3.2.2.4 Evaluation Metrics

- Accuracy (acc): Is usually defined as the percentage of correct predictions, which can be calculated by dividing the correct predictions (CP) with all the predictions (AP).

$$acc = \frac{CP}{AP} \tag{3.1}$$

- Loss: For this type classification problem with only two classes, binary cross-entropy is the default loss function. It calculates the actual difference between the predicted classes and the actual classes provided with the data.

- Precision (p): Measures the amount of samples that are relevant (true positives). This is calculated by dividing the true positives (TP) with the sum of true positives with false positives (FP).

$$p = \frac{TP}{TP + FP} \tag{3.2}$$

- AUC/ROC (AUC): Measures how well the model is able to distinguish classes, that is if the model correctly classifies the 1s as 1s.

$$AUC = \frac{SP - NP(NN+1)/2}{NP * NN} \tag{3.3}$$

where, SP is the sum of positive values while NP represents the number of positives and NN the number of negative values.

- Recall (r): Measures how many true positive samples are selected, by dividing the true positives with the sum of true positives and false negatives (FN).

$$r = \frac{TP}{TP+FN} \tag{3.4}$$

- F1: Measures test accuracy and it considers two major metrics to evaluate the score, precision and recall.

$$F1 = 2 * \frac{p * r}{p + r} \tag{3.5}$$

3.2.2.5 Results and Discussion

After training, the models were saved in a hdf5 format, which was specially useful for the evaluation process because with just a single line of code it is possible to load the model with all its features. To obtain a complete evaluation of the model, measuring the accuracy and loss values is not enough, we also have to look out for false positives. In the case at hand, having a model that identifies exoplanets with 99% accuracy is great but if the metrics that evaluate the number of false positives (F1, Precision and Recall) are significantly lower than that, then the model is not up to standard.

For that reason, the evaluation stage was performed using all the metrics explained in 3.2.2.4. The values presented in the tables below represent an average of 50 evaluations, all of them carried out after shuffling the test data to prevent memorization.

The reference values are represented in the first table. The best results described in [27] were obtained using a CNN and MLP architecture with two inputs each, global and local view. The values from [22] were obtained using CNN and LSTM models with the global view. Table 3.3 displays the results obtained using the models previously described in 3.2.2.3.

Table 3.2: Reference values for the Deep Learning models.

Classifier	Accuracy	Loss	AUC/ROC	Precision	Recall	F1
Shallue MLP [27]	0.94	-	0.98	-	-	-
Shallue CNN [27]	0.96	-	0.99	0.90	0.95	-
Dinis CNN [22]	0.94	0.24	0.97	0.94	0.94	0.94
Dinis LSTM [22]	0.93	0.24	0.97	0.94	0.94	0.94

Table 3.3: Resulting values from the ML classifiers and Deep Learning models using Approach 1.

Classifier	Accuracy	Loss	AUC/ROC	Precision	Recall	F1
SVM (Global)	0.84	5.43	0.79	0.52	0.69	0.67
Log.Reg (Global)	0.85	0.47	0.79	0.53	0.64	0.67
CNN (Global)	0.93	0.17	0.97	0.86	0.87	0.85
CNN (Local)	0.92	0.18	0.97	0.88	0.81	0.83
MLP (Global)	0.87	0.28	0.92	0.70	0.86	0.76
MLP (Local)	0.92	0.21	0.96	0.82	0.84	0.82
LSTM (Global)	0.93	0.18	0.97	0.83	0.92	0.85
LSTM (Local)	0.92	0.19	0.97	0.85	0.81	0.81

Overall, between both data inputs, the best results were obtained using the global view. For both CNN and LSTM the local view had better performance in the loss and precision metrics, but the difference is not significant enough and the global view offers greater values in four of the six metrics. Contrarily to those two, the MLP had significantly better results in the local view, this happened because usually less complex models perform better in smaller datasets. The local view has about 10x less samples than the global view, so it was expected for the more complex models to achieve slightly worse results.

In the first two rows of Table 3.3, the result for the ML classifiers is shown. Both are quite similar, having a decent accuracy result but lacking in all other metrics, specially in loss. The loss registered for logistic regression ends up being worse than all neural networks, but the loss obtained with SVM was extremely higher than usual. This value is justified because binary cross entropy was applied to all classifiers, which is a probabilistic distribution function that only works when the classifiers outputs a probabilistic distribution for the classes, SVM does not work that way so the value can become quite disproportional.

However, what stands out the most in this table is the difference between the results obtained by the neural networks for Accuracy/AUC and Precision/Recall/F1. The model performance had a significant decrease when testing for false positives, although having values between 0.85 and 0.90 can still be considered good results, the truth is that with the almost 0.10 difference between accuracy/AUC and those three metrics, the model can accurately identify an exoplanet most of the times but there is still a percentage of incorrect predictions and, as previously mentioned, a good model must perform well in all evaluation metrics.

In table 3.4 the results of the second approach (3.1.4.2) are displayed only for the global view. Considering that this view offered a overall greater performance, as shown in 3.3, it was decided that testing again with both views was unnecessary.

Table 3.4: Resulting values from the Deep Learning models using Approach 2.

Classifier	Accuracy	Loss	AUC/ROC	Precision	Recall	F1
CNN (Global)	0.90	0.27	0.93	0.77	0.76	0.75
MLP (Global)	0.88	0.32	0.90	0.65	0.74	0.68
LSTM (Global)	0.89	0.27	0.93	0.77	0.71	0.72

With the results displayed in 3.1.4.2 it is possible to affirmatively conclude that the so-called noise present in all light curves ends up being quite important for the networks. Smoothing the signal resulted in a notable decrease in the overall performance, making the steps taken in this second approach unnecessary.

The best performing model was obtained using the dual input functional CNN with Approach 1. Having two types of views allowed the network to gather a deeper understanding of the data and the difference between the results showed in Table 3.5 and all the above tables is quite notable.

Table 3.5: Performance of the dual CNN model using Approach 1.

Classifier	Accuracy	Loss	AUC/ROC	Precision	Recall	F1
CNN (Dual)	0.95	0.14	0.98	0.91	0.88	0.89

This, however, brings up a couple of questions: What patterns are these neural networks finding to identify exoplanets ? Why can't we look at those light curves in A.1 and correctly identify an exoplanet in all of them ? The next section will try to answer these questions.

3.2.3 Part 3 - Interpretable Machine Learning

The potential of machine learning is immense, it has the power to change a vast number of areas and it is increasingly becoming part of our everyday life. However, the major problem with these black box models is that most of the time they don't really explain their predictions. This is definitely something extremely worrisome because sometimes these predictions can influence decisions, and these decisions can impact people's lives. So the question is, how can we say without any doubt that this model's prediction results are valid ?

The main goal of this last part of the project is to answer this question, are the deep learning models obtained in the previous part correctly deciding what is and what is not an exoplanet?

Model interpretability is something relatively new to the field and its value is tremendous, in this case understanding what the model defines as exoplanet can not only validate (or not) the model predictions but may also help scientists to have a better understanding of what constitutes an exoplanet in the light curves obtained using transit method.

There are already a number of model agnostic methods that successfully explain model predictions. However, in this project, *Local Interpretable Model-Agnostic Explanations* (LIME) was the chosen method due to being probably the most influential of them all and also because it is able to explain individual predictions (hence the "Local" in the name).

Marco Tulio Ribeiro describes LIME in [25] as an algorithm that creates a type of substitute model, which is able to explain individual instances through approximations of the local predictions made by the model at hand. The idea behind this algorithm is to create a set of manipulated training data that once fed to the model it will train and then evaluate its prediction, allowing it to, in the end, figure out what variations correspond to a different prediction and, consequently, determine the features that contribute the most to this result.

The major challenge that comes with LIME is the local fidelity-interpretability trade-off, which means that a model should be complex enough to be interpretable to humans while maintaining the maximum fidelity possible to the given instance. This trade-off is defined in [25] as equation 3.6, where $\mathscr{L}(f, g, \pi_x)$ represents how inaccurate g is when, in the locality determined by π, trying to approximate f. To get the best of "both worlds" it is necessary to obtain the minimum values of $\mathscr{L}$ while still maintaining a lower complexity level Ω.

$$\varepsilon(x) = \underset{g \in G}{\operatorname{argmin}} \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{3.6}$$

The complexity of 3.6 only depends on the number of samples and the time taken to determine an individual prediction. For this project, using a CNN model it took about 30 minutes with 5000 samples which means that the model is quite complex and, as a result, it is likely that not all behaviours can be explained via representation.

The representation of the explanation entirely depends on the type of input data (image, text or tabular), which is unfortunate in this case because time series data is not currently supported. For that reason, a time series implementation of LIME called LIME For Time[5] was used.

3.2.3.1 LIME For Time

This is an implementation of LIME for time series data, created by Jonas Hering, Emanuel Metzenthin and Alexander Zenner. The idea behind this algorithm is the same as the original LIME, the main difference is how the explanation is represented and the type of replacement methods.

Because we are using data that is indexed in time order, and considering that the explanation should be easily understood by humans, this algorithm separates the signal into slices and the final result highlights the most important features in a graph, as shown in Figure 3.12.

To manipulate the data, it is created an equivalent of super-pixels (called here as slices) and in each variation one or more slices are replaced with one of three possible methods -explained in the following section-, thus becoming deactivated.

3.2.3.2 Implementation

The implementation of LIME for Time is quite easy and does not differ from the original LIME library. Only two functions are needed to obtain the results, *LimeTimeSeriesExplanation()* and *explain_instance()*. The former is the *LimeTimeSeries* class constructor and it receives as input a list of class names and the type of feature selection, which will determine how the algorithm will choose the N selection features.

[5]https://github.com/emanuel-metzenthin/Lime-For-Time

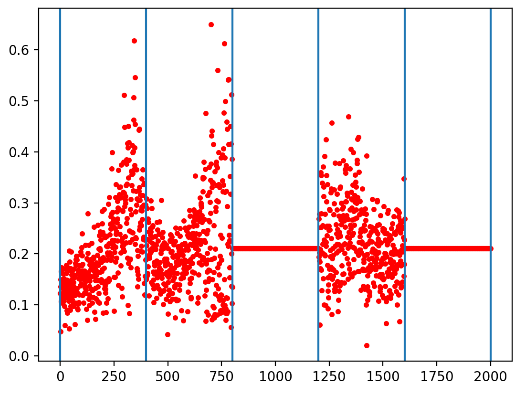


Figure 3.11: Representation of slices deactivation, using total mean, for a toy example in LIME For Time.

The latter returns a explanation for the prediction after receiving the instance to be evaluated, the training set and some characteristics for the explanation (the number of features, slices and samples). Additionally, it also allows the user to choose what type of replacement will be inserted into individual slices that are removed to make a prediction: total mean, random noise or the mean over a time period.

```
explainer = LimeTimeSeriesExplanation(
        class_names=['0', '1'], feature_selection='auto')

exp = explainer.explain_instance(series, new_predict,
   num_features=5, num_samples=5000, num_slices=40,
   replacement_method='total_mean', training_set=
   train_X_shaped, labels=(0, 1))
```

Note that, although in [25] a lasso path method is used, the feature selection in the first function is set as 'auto', which means that the algorithm will use a forward selection method that iteratively adds features. This method was chosen because it is the default method and the difference between this and lasso path is not significant. It is also important to take in consideration that the forward method is only used if the number of features is less than six, otherwise it will apply a highest weights method.

The returned explanation object is then transformed into a list and using *matplotlib* it is possible to create a figure like those presented in the following subsection.

However, the original Lime For Time did not support the neural networks created in the 3.2.2, thus some slight changes were needed. *Keras* needs *Numpy* arrays as inputs and this extension of LIME used *Pandas* dataframes. Additionally, the prediction function of *Keras* did not work, so a custom function was implemented.

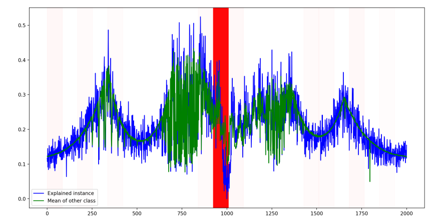
3.2.3.3 Results and Discussion

Both the explanation and the visual representation of the result entirely depend on the number of features and samples, increasing both values the result becomes more selective and thus improving its quality (up to a certain point). There is no formula to find the optimal combination of both values and, as such, trial and error becomes the more suitable approach.

Before looking at the explanation, it is important to understand what we are looking for. There are a couple of attributes that are expected to happen in a light curve signal when an exoplanet is present, but the most important are the periodicity and the short transit time leading to brief and sharp peaks. However, it is also important to have in mind that not all light curves defined as PC (or 1 in this case) contains an exoplanet appearance, thus becoming quite tricky to actually understand all signals provided (which becomes apparent when analyzing the light curves displayed in A).

Both figures in 3.12 represent an explanation provided by LIME For Time using 10 features and 5k samples. These two signals were chosen because they are complete opposites in terms of how easy it is to explain why there is an exoplanet. The first one has a uniform type of fluctuation but the dip in the middle immediately shows us that some object passed between the telescope and the target star, although the periodicity factor can not be considered because there is only one dip, being sharp and brief increases the probability of being an exoplanet. The explanation concurs with what it is expected and puts a darker red marker in the slice containing the dip.

The second figure is almost impossible to visually find a pattern that indicates the presence of an exoplanet. The explanation could be accurate or not, the truth is that it is not clear if an exoplanet is actually present despite being labeled as so. No attributes stands out, thus becoming indeterminate.



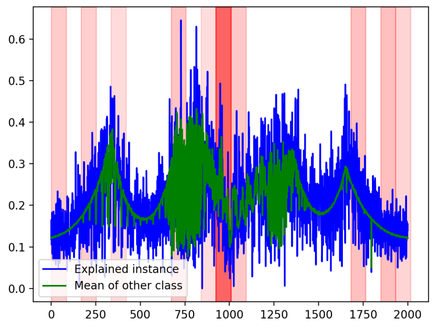


Figure 3.12: Explanation of two global view light curves.

However, this second figure displays something extremely important for the conclusion of the third and final part of the project. Notice that to the algorithm the slice in the center also contributes the most, despite being similar to the rest of the signal, indicating that some sort of bias is occurring. Analysing other signals and creating some experiments confirmed that not only there is a bias, but the explanation to why a signal is labelled as 0 or 1 is pretty much the same for all examples.

If, for example, the dip shown in the first figure is shifted to other section of the signal (switching those two sections), most of the time the explanation does not change to the new peak position, giving the same result as before.

With all these limitations, the explanations provided by Lime For Time can not be entirely trusted and, as a result, the interpretability for this data becomes inconclusive. Another mystery of the universe, it seems.

Chapter

4

Conclusion And Future Work

4.1 Main Conclusions

Exoplanet detection has been one of cosmology's most exciting endeavours for the last couple of decades. Analyzing large datasets provided by the *Kepler* mission with the help of machine learning, resulted in an increasing amount of planets being found outside of our solar system. This project had as its main goal creating deep learning models with an overall performance that would allow them to correctly find patterns in data corresponding to exoplanets.

As the project progressed, additional goals were added and creating neural network models became one of three parts. The first two, feature extraction and deep learning, demonstrated the differences between extracting features and training models with our own dataset and the patterns found by neural networks using the original data. The results were pretty clear, the latter resulted in a significant increase in overall performance when compared to the former. A set of models were tested and it was also found that the best results were obtained using a dual input functional CNN model. The final results did not entirely match the values described in both papers referenced for each part, although the difference wasn't so substantial.

Additionally, the second part also allowed the comparison between two data preprocessing approaches, one using the steps taken by the reference paper and other with an additional step, which removed the so-called noise present in all light curves. The results showed that this signal fluctuations played an important role for the neural networks' predictions, thus they should not be removed.

The third and final part of this project tried to answer the question of what attributes are the neural networks finding to declare a signal as containing an exoplanet. Using an extension of LIME, a library created to visually explain model predictions, the results showed some limitations and biases, thus becoming inconclusive.

All in all, this was my first introduction to large scale projects in machine learning and, in addiction to further increase my interest in the area, it allowed me to obtain a general perspective of how neural networks work and all the steps needed to prepare and then analyze data.

4.2 Future Work

The light curves analyzed in this project were obtained between the years of 2009 and 2013, however the Kepler space telescope continued to observe stars until 2018. As such, a possible future work would be to analyze the remaining data obtained by Kepler.

Kepler's work contributed immensely to our knowledge of the universe and, despite having already stopped its job, there is already a replacement called *TESS* ([2]). Hopefully in the future we can get access to data on new stars and, who knows, identify more exoplanets.

In terms of improvements, the overall performance of the models in 3.2.2.3 weren't quite as good as expected, lacking specially in the false positives metrics (for both CNN and LSTM models). For that reason, more work should be done to increase these values. The use of ensemble methods, which combines multiple algorithms to achieve better results, is a possible solution to this problem.

Finally, the inconclusive result of the third part of this project and the lack of sources to interpret models with time series data, highlights the importance of having an algorithm that is actually able to provide accurate results for neural network models. As of today, a model agnostic explanation for time series data named LEFTIST[1] was introduced but no public available implementation was found.

[1]https://ieeexplore.ieee.org/document/8995349

Appendix

A

Appendix A

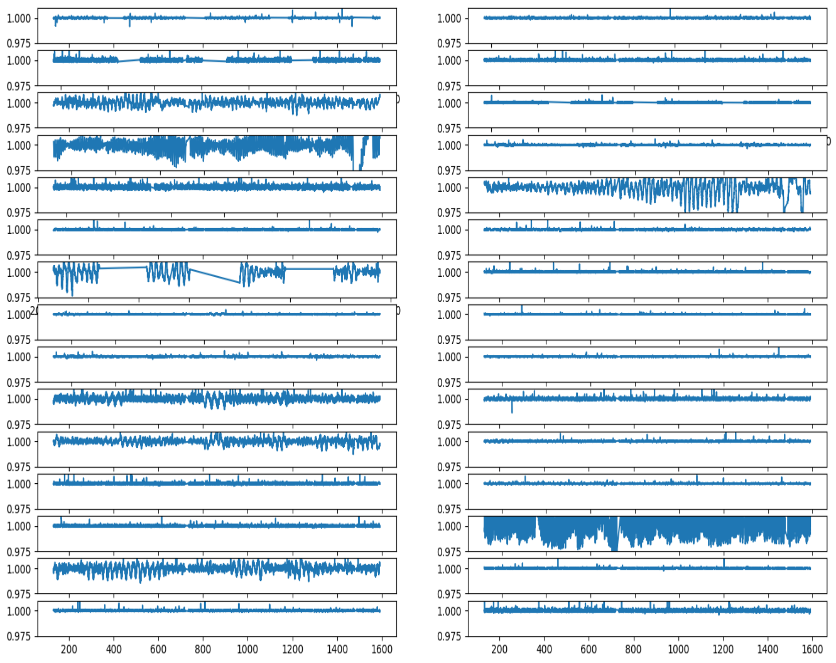


Figure A.1: Set of PC light curves with scaled PDCSAP FLUX in relation to time.

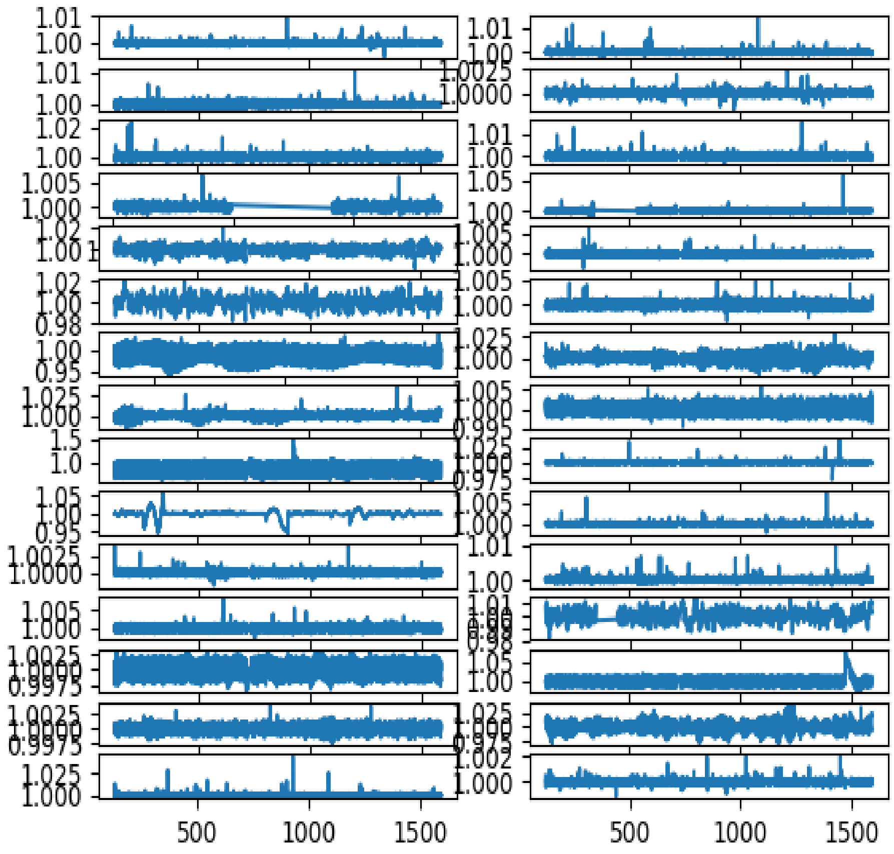


Figure A.2: Set of NTP light curves with scaled PDCSAP FLUX in relation to time.

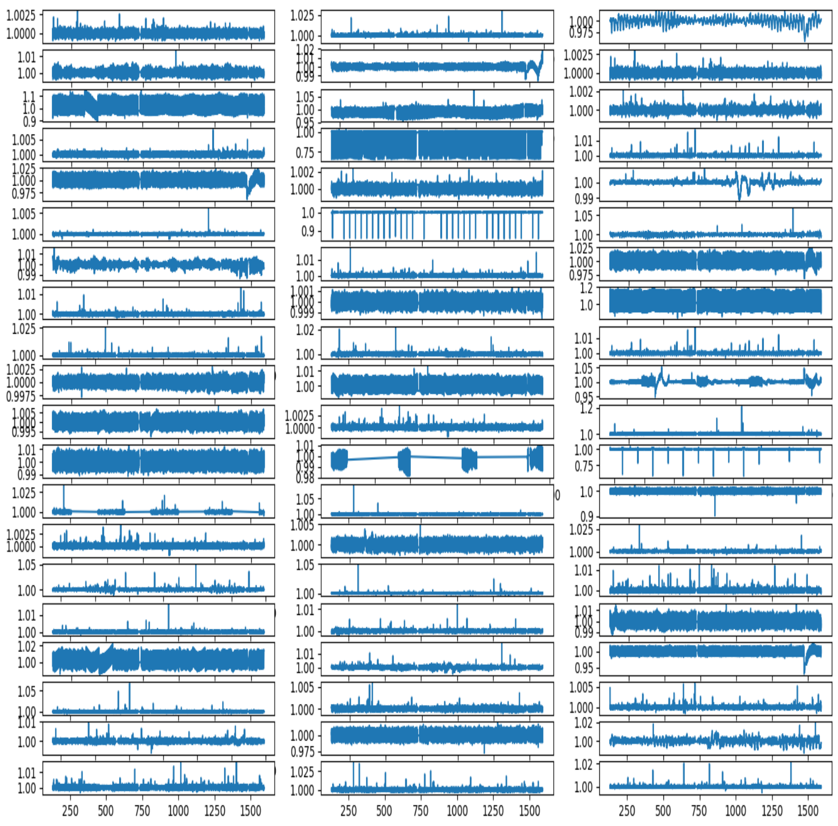


Figure A.3: Set of AFP light curves with scaled PDCSAP FLUX in relation to time.

Bibliography

[1] A Gentle Introduction to Pooling Layers for Convolutional Neural Networks. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks. Accessed: 2020-02-16.

[2] Exoplanets: Worlds Beyond Our Solar System. https://www.space.com/17738-exoplanets.html. Accessed: 2020-02-07.

[3] FITS File Handling. https://docs.astropy.org/en/stable/io/fits/. Accessed: 2020-01-29.

[4] FITS(Flexible Image Transport System). https://fits.gsfc.nasa.gov/. Accessed: 2020-01-28.

[5] Kepler, the Little NASA Spacecraft That Could, No Longer Can. https://www.nytimes.com/2018/10/30/science/nasa-kepler-exoplanet.html. Accessed: 2020-02-07.

[6] Logistic Regression — Detailed Overview. https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc. Accessed: 2020-02-14.

[7] Machine Learning Basics with the K-Nearest Neighbors Algorithm. https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761. Accessed: 2020-02-14.

[8] Machine learning style preprocessing with lightkurve¶. https://docs.lightkurve.org/tutorials/05-advanced_patterns_binning.html. Accessed: 2020-05-19.

[9] Microlensing. https://www.planetary.org/explore/space-topics/exoplanets/microlensing.html. Accessed: 2020-02-07.

[10] Multi-Extension FITS File Format. http://www.stsci.edu/itt/review/dhb_2011/Intro/intro_ch23.html. Accessed: 2020-01-29.

[11] Radial Velocity. https://www.planetary.org/explore/space-topics/exoplanets/radial-velocity.html. Accessed: 2020-02-07.

[12] Support Vector Machine — Introduction to Machine Learning Algorithms. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. Accessed: 2020-02-17.

[13] The Goldilocks Zone. https://science.nasa.gov/science-news/science-at-nasa/2003/02oct_goldilocks/. Accessed: 2020-02-07.

[14] Top 9 Algorithms for a Machine Learning Beginner. https://towardsdatascience.com/top-10-algorithms-for-machine-learning-beginners-149374935f3c. Accessed: 2020-02-14.

[15] Transit Photometry. https://www.planetary.org/explore/space-topics/exoplanets/transit-photometry.html. Accessed: 2020-02-07.

[16] Understanding LSTM Networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2020-03-23.

[17] Understanding RNN and LSTM. https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e. Accessed: 2020-03-22.

[18] What Is Deep Learning? https://machinelearningmastery.com/what-is-deep-learning/. Accessed: 2020-02-14.

[19] Zero Padding in Convolutional Neural Networks. https://deeplizard.com/learn/video/qSTv_m-KFk0. Accessed: 2020-02-15.

[20] Abraham Botros. Artificial Intelligence on the Final Frontier: Using Machine Learning to Find New Earths, 2014. [Online] https://pdfs.semanticscholar.org/8c96/1f7357fbc3e22e0e5417744af24662222818.pdf. Accessed: January 30, 2020.

[21] Rachel Bowens-Rubin. Feasibility-study for space-based transit photometry using mid-sized nanosatellites. https://www.semanticscholar.org/paper/Feasibility-study-for-space-based-transit-using-Bowens-Rubin/b51db0a153ccb65c9a68a1f02d105f42a5d64531. Accessed: 2020-02-07.

[22] Dinis Marques Firmino. Exoplanet Transit Detection using Deep Neural Networks, 2018. [Online] `https://github.com/dinismf/exoplanet_classification_thesis/blob/master/reports/dissertation.pdf`. Accessed: February 14, 2020.

[23] Yarin Gal and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout inRecurrent Neural Networks, 2016. [Online] `https://arxiv.org/pdf/1512.05287.pdf`. Accessed: April 16, 2020.

[24] Xiang Wang Kai Yang, Zhitao Huang and Xueqiong Lo. A Blind Spectrum Sensing Method Based on Deep Learning. `www.researchgate.net/publication/333154420_A_Blind//_Spectrum_Sensing_Method_Based_on_Deep_Learning`. Accessed: 2020-02-14.

[25] Sameer Singh Marco Tulio Ribeiro and Carlos Guestrin. Why Should I Trust You?": Explaining the Predictions of Any Classifie, 2016. [Online] `https://arxiv.org/abs/1602.04938`. Accessed: May 20, 2020.

[26] Wojciech Zaremba Rafal Jozefowicz and Ilya Sutskever. An Empirical Exploration of Recurrent Network Architectures, 2015. [Online] `http://proceedings.mlr.press/v37/jozefowicz15.pdf`. Accessed: April 16, 2020.

[27] Christopher J. Shallue and Andrew Vanderburg. Identifying Exoplanets With Deep Learning: A Five Planet Resonant Chainaround Kepler-80 And An Eight Planet Around Kepler-90, 2018. [Online] `https://www.cfa.harvard.edu/~avanderb/kepler90i.pdf`. Accessed: January 30, 2020.

[28] Xiaolin Hu Xiang Li, Shuo Chen and Jian Yang. Understanding the Disharmony between Dropout and Batch Normalization byVariance Shift, 2018. [Online] `https://arxiv.org/pdf/1801.05134.pdf`. Accessed: April 16, 2020.

[29] Yoshua Bengio Yann LeCun and Geoffrey Hinton. Deep Learning. `https://doi.org/10.1038/nature14539`. Accessed: 2020-02-14.