Search for Exoplanets

Petra Brčić
Applied Mathematics
petrabrcic94@gmail.com

Sandro Lovnički
Computer Science
lovnicki.sandro@gmail.com

June 30, 2018

Abstract: With the launch of Kepler space observatory in March 2009, the search for exoplanets has extensively began. There are currently 3,758 confirmed exoplanets and contrary to the early approaches of validating exoplanet candidates by hand, machine learning has begun taking its toll on the subject. We investigate a convolutional neural network model for detecting transiting exoplanets from Kepler observatory light-curve data collected over a period of 4 years, with over 150,000 documented stars. On April 18, 2018, a new satellite (TESS) has been launched, specificaly designed to search for exoplanets using the transit method. After it's 2-year planned mission, complete data should be available to public and more than 20,000 new exoplanets are expected to be found.

Keywords: Kepler, light-curve, exoplanet, machine learning, convolutional neural network.

Contents

I. Introduction

We investigated an implementation of convolutional neural network AstroNet proposed by Shallue and Vanderburg, built with TensorFlow. Using it's high level syntax, we dug deeper into workings of TensorFlow as well as AstroNet, finally building our own classification model with many additional useful inner logs (like mid-evaluation filter values plotting and convolutional layers architecture analysis).

In this paper, we will be talking about processing the raw Kepler light-curves so it can be fed to the neural network in a more meaningful form and overall architecture, learning process and power of our convolutional neural network model. Lastly, we present some interesting results we encountered during this search for a new home.

II. Data

Data used for this project is obtained by combining NASA Exoplanet Archive data for Threshold-Crossing Events which (currently) consists of 34032 documented TCEs, each having 155 column features and raw Kepler light-curves (star's flux (brightness) over time) at Mikulski Archive for Space Telescopes consisting of over $150,000$ star light-curves monitored by Kepler.

Most important TCE features for our projects are `kepid`, `tce_period`, `tce_time0bk`, `tce_duration` and `av_training_set` which (respectively) correspond to the Kepler identifier of host star, period of the event, time of the first appearance (more precisely; time of the beginning of drop + `tce_duration`/2), duration of the event (drop in star's brightness) and label from set $\{PC, NTP, AFP, UNK\}$ meaning planet candidate, non-transit phenomena, astrophysical false positive and unknown. Those features will enable us to download just the Kepler light-curves of interest and later on to process each light-curve to extract and amplify features of TCE.

i. Creating Dataset

At the time of our data collecting, and older version of a TCE table was available with 20367 labeled TCEs. Distribution of `av_training_set` labels is as follows:

- 3600 TCEs with label PC (planet candidate)
- 2541 TCEs with label NTP (non-transit phenomena)
- 9596 TCEs with label AFP (astrophisical false positive)
- 4630 TCEs with label UNK (unknown)

We used that table, specifically `kepid` colum to download just the light-curves for those stars from Kepler data. These light-curves are taking about 90GB of space and are just a fraction of all the Kepler data.

Still, our machines are not fully prepared to tackle the preprocessing and learning from such a vast amount of not at all simple data, so we decided to choose randomly just some of the TCEs from table. We ended up selecting 1058 TCEs labeled as planet candidate, 471 TCEs labeled as astrophisical false positive and 734 TCEs labeled as non transit phenomena.

As an example, here is a small table of selected features of one TCE documented for a star with `kepid` 5088591:

kepid	5088591
tce_period	14.5324
tce_time0bk	140.725
tce_duration	0.13358333333333333
av_training_set	PC
...	...
tce_plnt_num	1
tce_eqt	665.0
...	...

ii. Preprocessing Lightcurves

In order to feed our model, we first need to process the raw light-curve into more useful and compact form. Raw light-curve is loaded as an array of smaller arrays called quarters which represent distinct periods in Kelper telescope's configuration, position, etc. We thus

might expect quarters to be on a slightly different scale. In figure 2 we show lightcurve for `kepid` 5088591 right after loading `time` and `flux` arrays, as well as after dividing them by median quarter values.

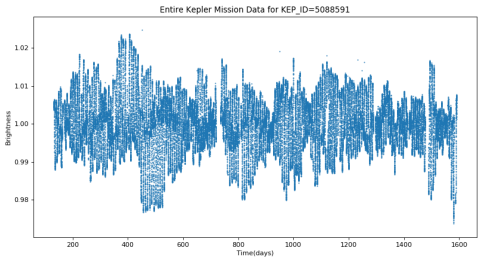

Figure 1: *Raw light-curve data*

It is easy to see that almost every star has an unpredictable variation in its flux, so the data is pretty distorted and irregular which is something that is to be avoided because it would interfere with our model's conclusions and feature extraction. Thus, we calculate a B-spline over the entire light-curve data, ignoring points that cannot be fitted or are of type `None`. This spline can be seen in figure!3 colored red.

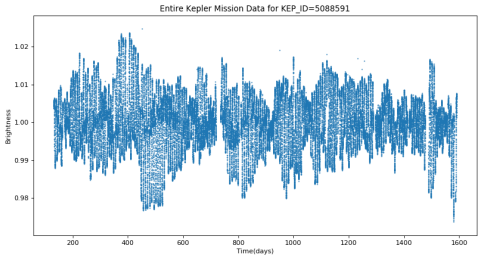

Figure 2: *Raw light-curve data with fitted B-spline*

Next, we divide light-curve by the spline to obtain 4. Notice how specific TCEs can now be seen without even enlarging.

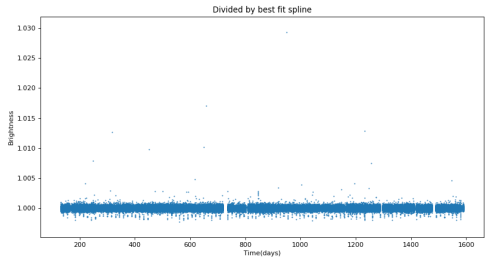

Figure 3: *Raw light-curve data divided by spline*

Now we need to exactly find the position of desired TCE which we know from TCE table, column `tce_time0bk`. Focusing just on that segment of this plot, we get figure 5

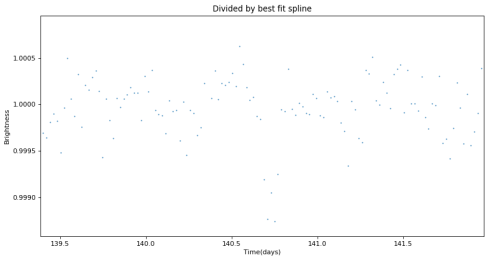

Figure 4: *Enlarged area of the TCE*

We see that the amount of useful data of this "drop" is very poor, but we must remember that those events are periodic by definition so we hope that there is more instances of that same "planet" crossing during the lifetime of Kepler. We find the `tce_period` from TCE table and fold every drop separated by `tce_period` to `time0bk`. Now we divide TCE's duration into 201 discrete points and calculate median value for flux in each of 200 intervals in between. This results in a much richer, and uniform for all events, representation of a TCE which can be seen in!6. We call that view a "local view" and it is going to be the input of our model.

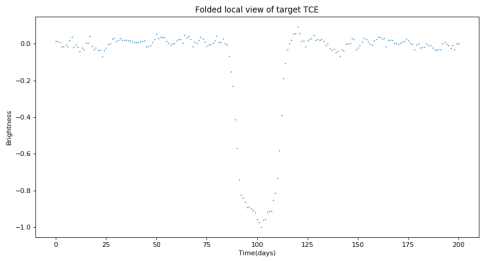

Figure 5: *Folded local view of the TCE*

III. Convolutional Neural Network

With the recent rise of neural networks, many variations are being explored for specific problems. In classification, convolutional neural networks have been proven to yield best results and we will be using one in this project. Let's first briefly introduce the reader with the concept of convolutional neural networks and how exactly they differ from regular neural networks.

A convolutional neural network (CNN) can be viewed as a deep neural network with addition of feature extraction mechanism in the from, that is — before the classification with fully connected layers begin. Building block of convolutional neural network are convolutional layers with pooling and fully connected layers for classification. General scheme of one CNN with two layers of 2D convolutions can be seen in figure 7.

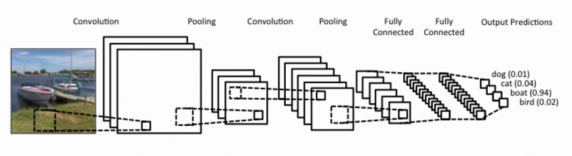

Figure 6: *Scheme of a 2D CNN*

In previous figure, we have a CNN that is used for classifying 2D data (generally, images are third order tensors, but we can assume image to be grayscale). Our data is not two-dimensional so we will be using 1D convolution with ReLu function as activation and maximum pooling, each of which will be explained in the following subsections, specifically as they were in out project.

The full graph of our convolutional neural network viewed in TensorBoard can be seen in figure 8.

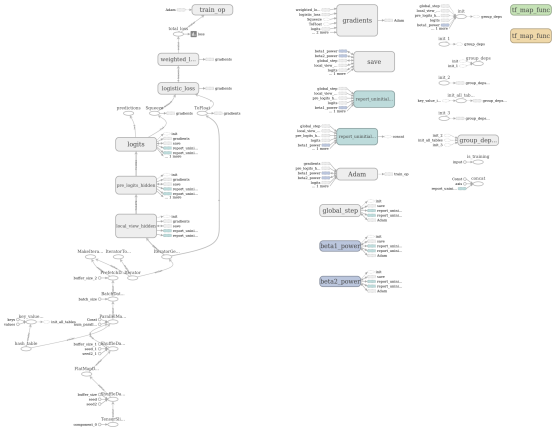

Figure 7: *TensorFlow graph of our exo_nn convolutional neural network*

i. Convolutional Layer

A convolutional layer consists of neurons that are not fully connected, that is — each neurons in this layer is connect to just the small portion of the input. The convolution layerâĂŹs parameters consist of a set of learnable filters. During the forward pass, we slide (more precisely, convolve) each filter across the input and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the the input, we will produce a 1-dimensional activation map that gives the responses of that filter at every position. Furthermore, we can stack multiple convolutional layers before pooling their output. One thing to notice here is that there are two options in doing the described procedure; with padding the input and without padding it, meaning that with padding we add zero values to the left or right of the input so the convolution step on the first/last few inputs in the input array will be valid (there will be enough inputs on each

side to do the dot product). We use padding, thus are able to obtain same-size output after the convolution. The convolutional kernel size we use is 5.

We have a set of 16 filters in first convolution block and each of them will produce a separate 1-dimensional activation map. We stack these activation maps along the depth dimension and produce the output. So, for out input representation — arrays of length 201 (shape $(?, 201)$, where question mark is batch size during training and 1 during prediction), the output of convolution layer will be of shape $(?, 201, 16)$. We then feed this data into another convolutional layer with a set of 16 filters yielding as outputs tensors of shape $(?, 201, 16)$.

Also, there are 32 filter in the second convolutional block taking as input the output of pooling the outputs of the first convolutional block. The output of this second convolutional layer is the set of features our convolutional blocks have learned and it is then reduced more with maximum pooling, flattened to shape $(?, 1472)$and passed to fully connected layers for classification.

ii. Rectified Linear Unit

For our activation function, we used rectified linear unit (ReLu). It is applied to the dot product of filter and current area the neuron is "looking" at, to produce an output value. One other valid choice for this task would be the sigmoid, but it would not stress enough the importance of larger values.

To be specific, when our network is trained for 1000 steps with sigmoid activation function, it only manages to get accuracy of 0.5265 while with ReLu it goes to 0.9735 in the same number of steps. Furthermore, sigmoid model has 0 true positives in the confusion matrix and we present why exactly that is so in the next figure.

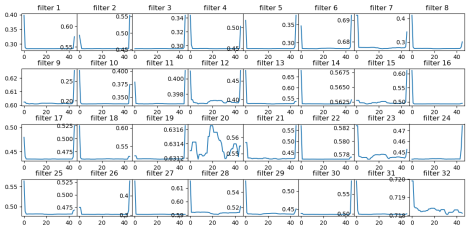

Figure 8: *Output of 32 filters after last pooling before FC (using sigmoid activations)*

As we can see in figure 9, almost no features are detected and therefore almost every example will be classified as non-transiting phenomena. Notice how vertical scale is pretty much the same and this is because of pretty much the same outputs from sigmoid. We also present the same picture for ReLu.

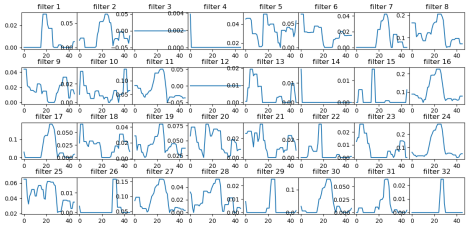

Figure 9: *CHANGE IT Output of 32 filters after last pooling before FC (using ReLu activations)*

iii. Max Pooling

Pooling is a method of downsizing the input data, but keeping the structure. It is used to abstract features obtained by convolutional layer into more complex features that then go further into neural network. We used max pooling with size 7 and stride 2 which lead to reducing the output size of the first convolutional block from 201 to 98 and output size from the second convolutional layer from 98 to 46. Max pooling is done almost like convolution, but we do not calculate the dot-product of "pool filter" of length 7 with the part of the input of length 7. Instead, we just take maximum value of 7 input values we are looking at, slide 2 steps, collect a maximum value again and so on.

iv. Fully Connected

After our second convolutional block, we ended up with 32 values of length 46 which are now flattened to form a 1472 inputs for a network of 2 fully connected layers with 1024 neurons each. As activation function, we set ReLu.

We also tried with 4 layers with 512 neurons in a layer (and many other combinations) which produced similar or worse accuracy.

v. Sigmoid Output

Lastly, the computations of fully connected layers are sumarized in a single output value representing the probability that a TCE is a planet candidate. Therefore, we use a sigmoid function as activation for this last neuron.

IV. Training

We train a convolutional neural network model on a randomly constructed set of data consisting of 1058 planet candidates, 471 astrophysical false positives and 734 non-transiting phenomena, summing to 2263 data examples. For this model (convolutional neural network described above), astrophysical false positives and non-transiting phenomena are both mapped to label 0 and planet candidates are labeled 1.

i. Parameters

We shuffle the data and partition it into a training set consisting of 1810 examples, a validation set of 226 examples and a test set of 227 examples. Training is done with batch size of 16, in two main parts; first we train for 3000 steps with learning rate 10^{-5}, and then we try to fine tune the accuracy with learning rate 10^{-6} for another 1000 steps. Optimization in both parts is done with Adam optimizer.

ii. Progress

The training process for parameters described above can be most easily viewed through TensorBoard interface for visualizing computation graph and its execution.

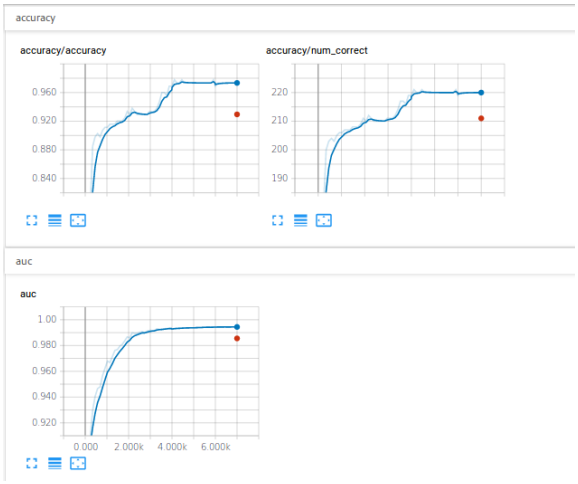

Figure 10: *Accuracy and Area Under Curve of our model during training and testing, viewed from TensorBoard*

The accuracy starts stagnating at around 0.9735 and it seems like we can not get rid of some of the false predictions. View of the values of the confusion matrix during training steps and test can be seen in figure 12.

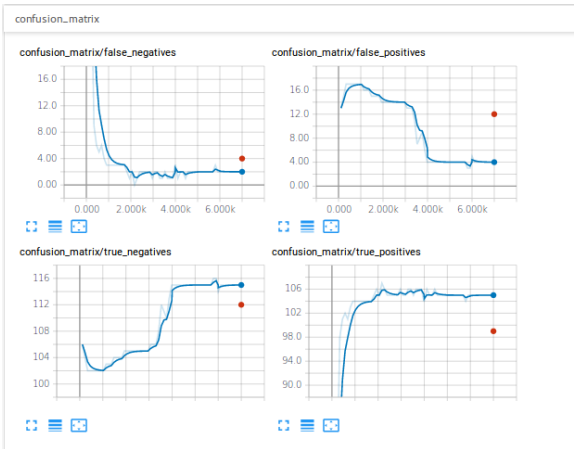

Figure 11: *Confussion matrix values during training and testing*

To conclude, after 7000 steps we obtained an accuracy of 0.9734513 with 220 correctly classified examples in the validation set (6 wrongly;

2 false negatives and 4 false positives). Total training time was about 16 minutes and 20 seconds, on a very slow machine.

iii. Testing

Testing results can also be seen in figures 11 and 12 and are represented with a red dot. Explicitly, we obtained a 0.9295154 accuracy on the training set with 211 correctly classified examples (16 wrongly; 4 false negatives and 12 false positives).

It is expected that there are more false positives because they are tricky to find even for professionals with specialized equipment.

V. Results

In this last section, we present some of the most interesting results we encountered during our journey of training and testing our exoplanet classifier.

i. The weirdest

ii. The cleanest

iii. The unknowns

References

[1] https://en.wikipedia.org/wiki/Kepler_(spacecraft)

[2] https://en.wikipedia.org/wiki/Transiting_Exoplanet_Survey_Satellite

[TESS] https://tess.mit.edu/

[3] https://en.wikipedia.org/wiki/Methods_of_detecting_exoplanets

[4] https://en.wikipedia.org/wiki/List_of_exoplanets_detected_by_microlensing

[5] https://en.wikipedia.org/wiki/List_of_exoplanets_detected_by_radial_velocity

[6] https://en.wikipedia.org/wiki/List_of_exoplanets_discovered_using_the_Kepler_spacecraft

[Shallue and Vanderburg, 2017] Christopher J. Shallue, Andrew Vanderburg (2017). Identifying Exoplanets with Deep Learning: A Five Planet Resonant Chain around Kepler-80 and an Eighth Planet around Kepler-90 arXiv:1712.05044 [astro-ph.EP]

[Mikulski Archive for Space Telescopes] https://archive.stsci.edu/

[NASA Exoplanet Archive] https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=q1_q17_dr24_tce

[TCEs table] https://exoplanetarchive.ipac.caltech.edu/cgi-bin/TblView/nph-tblView?app=ExoTbls&config=q1_q17_dr24_tce