MLND_Report

February 10, 2018

1 Machine Learning Engineer Nanodegree

1.1 Capstone Project

Patrick Daskas February 5, 2018

1.2 I. Definition

1.2.1 Project Overview

This project uses a convolutional neural network to determine if a star analyzed by the Kepler Mission has planet candidates in orbit.

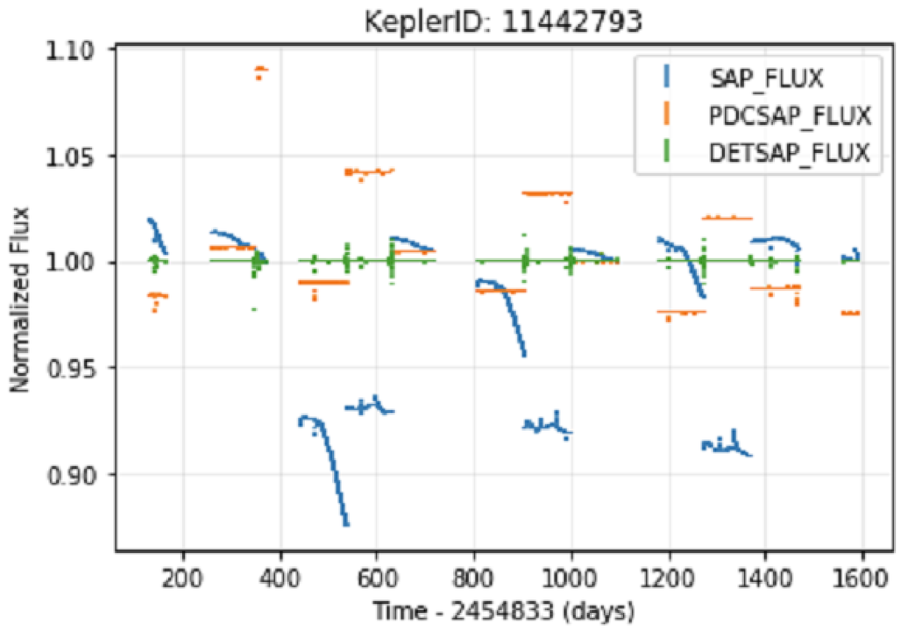
1.2.2 Problem Statement

The amount of data acquired by the Kepler telescope is enormous. How are exoplanets detected in all the data? Almost every star observed has over 4000 data points per quarter. There are approximately 70000 data points that describe the change in brightness and the time in which it occurred for each star. Add to that over 100000 stars with this type of data, and one can quickly realize that manual detection of exoplanets seems absurd. There are websites today that pull this data and have a userbase analyze each of the image sets. They are attempting to manually categorize stars as having or not having exoplanets.

I built a convolutional neural network to automatically classify targets in the dataset. I constructed images as inputs to the network and the model learned to classify images that have planet candidates from those that do not. These images graph the change in brightness for each target across time.

1.2.3 Metrics

I elected to balance data sets so that there exists equal planet candidates and equal false positives. I will be looking at loss and accuracy to evaluate the model. I will graph the history of the accuracy, loss, validation accuracy, and validation loss across several epochs. Overall, I will look to reduce the validation loss and increase the validation accuracy. Using Keras, I will also use the evaluate function against my test data to determine its respective loss and accuracy.



k90_stitched_flattened.png

1.3 II. Analysis

1.3.1 Data Exploration and Visualization

The raw data from the Kepler Mission for a star, in this case Kepler-90, is represented by the PDCSAP_FLUX. The PDCSAP_FLUX is a result of NASA's extensive analysis pipeline of Kepler Mission data. It tries to remove anomalous data such as outliers and effects from instruments onboard the Kepler telescope. The process also tries to minimize the impact of other astronomical phenomenon.

Figure A shows the observed normalized brightness at a given time during its observation. This star has 8 planets orbiting it but there is hardly any indication of a planet in looking at the image. In order to start observing transits would require more data preprocessing and higher resolution images.

Later on, in the Data Preprocessing section, I will discuss the DETSAP_FLUX which is a result of my data preprocessing.

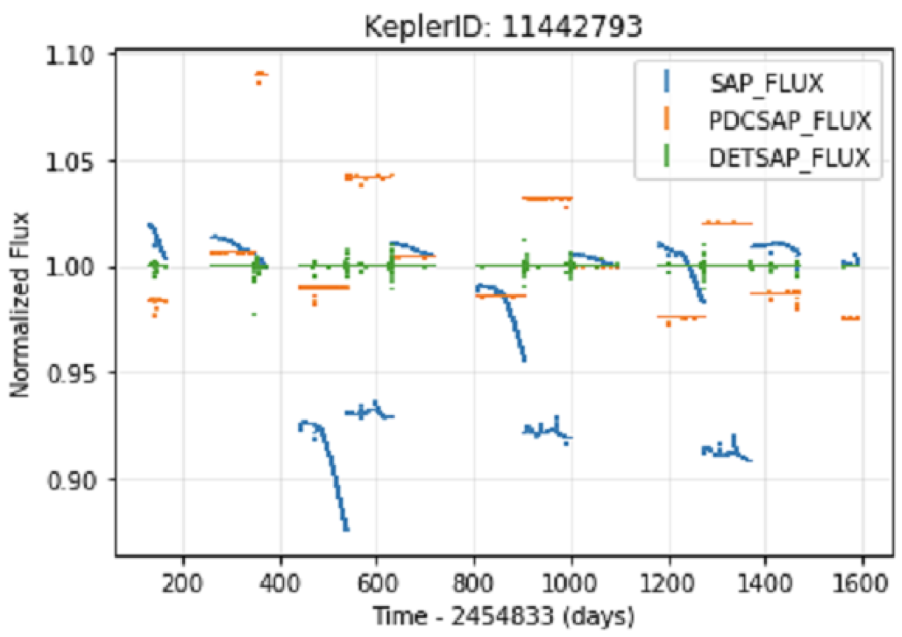
1.4 III. Methodology

1.4.1 Data Preprocessing

The raw data I used to create the inputs was for approximately 4400 Kepler targets. Initially this data came out to be close to 40 gigabytes of data, Acquiring this classified data required the execution of wget scripts and several manual downloads of the necessary files.

For the planet candidate targets, I took data from http://archive.stsci.edu/missions/kepler/lightcurves/tarfi
I downloaded each long cadence tgz file (_long.tgz) for all 17 quarters. Each contains a collection of fits files for various stars that were scanned in that quarter.

For the false positives, I executed wget scripts by quarter from http://archive.stsci.edu/missions/kepler/lightcurves/tarfiles/wget_scripts/. These scripts



k90_stitched_flattened.png

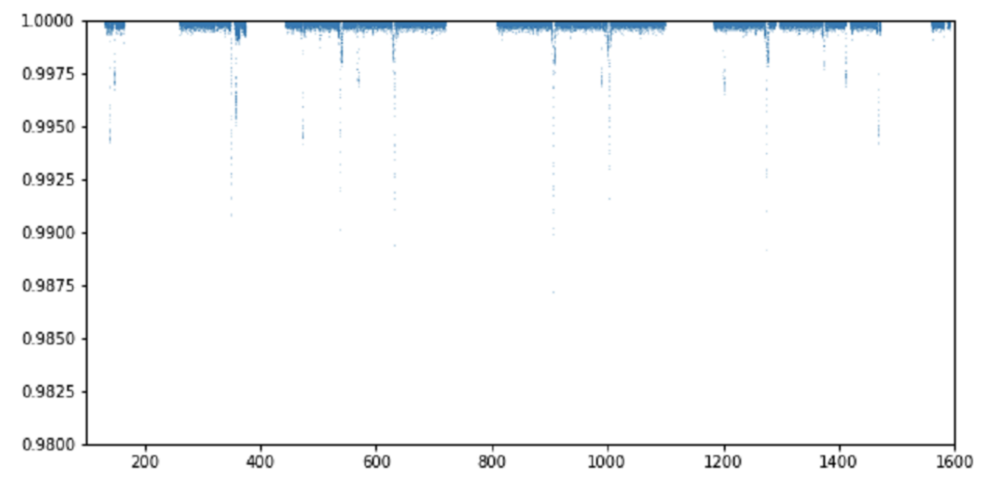
downloaded long and short cadence files. To be consistent with the Exoplanet KOI data, I removed all short cadence files (these files ended in sc.fits) and kept the long cadence files (ending in lc.fits).

Using PyKE (https://github.com/KeplerGO/pyke), I stitched together 17 quarters of data for each Kepler target. The data was normalized using PyKE kepflatten function. Normalization the flux is an expensive operation and I split the work across multiple cores on the CPU. This process took about 120 hours to complete.

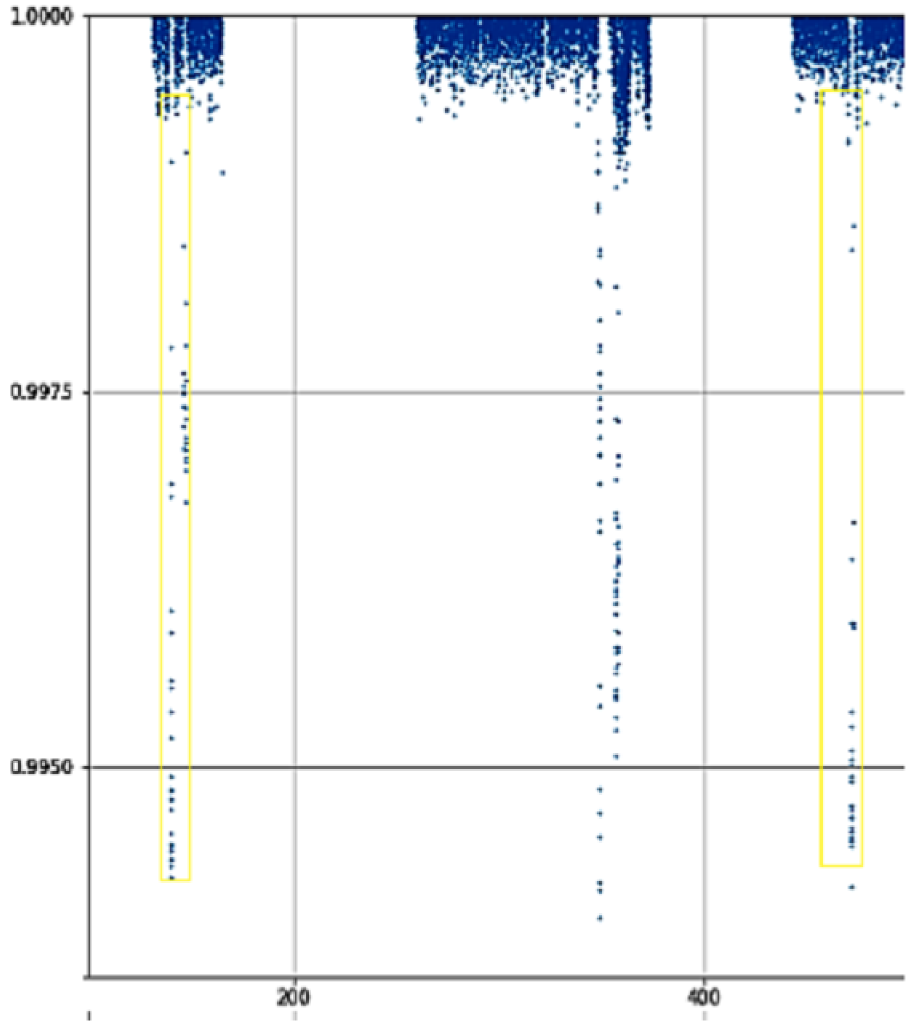
The normalized fits file contains datapoints that represent a normalized brightness and time. Using this data with matplotlib, I created a picture of the change in brightness for a given star across 17 quarters. I removed outliers in the y-axis from the image by setting a boundary on the plot to only keep data from .980 to 1.000. When there is no transiting planet, the brightness is very close to 1.000 and when large planets transit a star the brightness could reduce by up to 2%.

Revisiting the prior image, we can see the green line which shows the DETSAP_FLUX. This is the normalized brightness after using the kepflatten function.

The picture that I created for each Kepler target with matplotlib takes the data from the DETSAP_FLUX and graphs it over time.



Looking more closely at this image can reveal transitting planets. The first time a planet transits in front of the star is on day 141 and then again on day 472. The next two transits should occur on day 803 and day 1134, but during these times, the Kepler telescope did not capture any data.



The picture created for each target has a resolution of 432 x 3600 (height x width). This is needed to capture small variations in the data (where the brightness dips minimally for smaller planets). This large image, as expected, created memory issues even with the use of a generator function.

During training further preprocessing was done on these inputs on the fly. I used a Keras ImageDataGenerator to reduce the image by 50%. I also converted the image to grayscale to reduce the complexity of the data that would be used for training. This was done since color had no significance on the images I created.

My set contained about 2200 planet candidate targets, and 2200 false positive targets. 80% of the data was used for training. The remaining 20% was used for training. 33% of the training data was used for validation.

1.4.2 Implementation and Refinement

Initially my png files were 432 x 3600. I wanted a higher resolution to capture the details as the graph for each kepler id contained roughly 70000 points. Due to memory constraints on the GPU (6GB memory), I was forced to scale down the image.

Using the ImageDataGenerator class, I set the target_size to be 216x1800 and set my batch size to 4. I set the color_mode to grayscale, as color is not relevant in the images. The only relevant features are datapoints.

With the inputs finalized, I constructed a simple CNN. The CNN contained two 2D convolutional layers, each with a relu activation. Following these layers were a GAP layer, and a fully connected layer with a softmax activation. This resulted in 9570 trainable parameters. I compiled the model using the 'adam' optimizer and selected the categorical_crossentropy loss function. I setup a checkpointer to save the weights of the best model during training.

Training my initial CNN showed very little increase in the validation accuracy after 5 epochs. I increased the batch size, reduced the image size (216 x 1800 with a batchsize of 4) and let it run for 75 epochs. It was not until 60 epochs that the accuracy started to increase and stabilize. The validation accuracy achieved was 99.55%. Evaluating the model against the test data resulted in 99.2% accuracy.

Adding maxpooling layers increased the loss significantly and the accuracy did not increase. In the end I created a deeper model by adding an addition convolutional layer which resulted in significantly less epochs (18) to train the model up to 99%+.

1.5 IV. Results

1.5.1 Model Evaluation, Validation, and Justification

The first model had an accuracy of 99.2% when evaluated against new unseen data. After 75 epochs though, I believe this model would hae reached 100% test accuracy which raises some questions. Even the final model performance's against my test data showed an interesting result. The test accuracy reached 100%.
Perhaps this is a a result of insufficient data. If my training model identified a set of features, and my validation data only had a subset of the features, then this would support the 100% validation accuracy. The same case exists for the test set. To build a more robust model would require a larger set of data, or better image resolution. Because of hardware constraints, I was forced to reduce the resolution of my images, making it harder for the CNN to detect more patterns in the images.

To an extent I believe this solution is significant enough to have solved the simple classification problem. I think that if I could find a way to load in higher resolution images without memory issues, then this model would be more trustworthy. I also believe with better preprocessing and hardware this model could be enhanced to determine the number of planet candidates orbiting a star.

1.6 V. Conclusion

1.6.1 Reflection and Improvement

I started this project to see if a neural network could help classify stars as having planets. This proved to be quite challenging. Understanding and acquiring the raw data (40GB) required quite a significant amount of time. Flattening the data was another expensive operation that took 5 days of CPU time. With the inputs finalized, I looked at several images and compared them to the findings in each NASA Data Validation Report. Distinguishing planets from false positives was not easy, but I thought a CNN would be able to detect the patterns that define planet candidates from false positives. In the end, I was forced to sacrifice image quality because of GPU memory limitations, which I thought removed crucial information required for the CNN to classify the data. Still though, the CNN learned the features and classified the data correctly 99.2+% of the time.

I think with a larger dataset and higher resolution images would result in a more accurate model. Additionally, I think understanding more of the details in the paper https://arxiv.org/pdf/1712.05044.pdf could have helped me build a more trustworthy model. In that paper the authors built a CNN with two sets of inputs which I was not quite sure on how to implement in Keras. On page 8 of the paper their model shows the two separate layer stacks that are connected the final four fully connected layers. They also used a 1D convolutional layer which I did try but again ran into memory constraints. I think with more time I could have adjusted my hyperparameters to overcome the memory constraints. In section 3.3 of the paper they also discussed folding their light curve on the TCE period and then binning it to produce a new vector. To do this would require that they separate out the light curves as a result of each planet around a target star. I did not know how to accomplish this, and it would likely have taken more time to preprocess my data.