Exoplanet Hunting: Kepler Labeled Time Series Data Analysis

Group 8

Reported by

Zhilin Wang
Zixuan Huang
Renee Adonteng

May 1, 2020

Table of Contents

1. Introduction 2
2. Problem Statement 2
3. Description of Data 2
3.1. Source of Data 2
3.2. Overview of Dataset 3
4. Data Preprocessing 3
4.1. Data Scaling 3
4.2. Outlier Removing 4
4.3. Data Resampling 4
4.4. Data Relabelling 4
5. Math Background 5
5.1. Loss: binary__crossentropy 5
5.2. Evaluation: F1 6
6. MLP 7
6.1. Model Structure 7
6.2. Experimental Setup 7
6.3. Results 8
7. CNN 10
7.1. Model Structure 10
7.2. Experimental setup 11
7.3. Results 11
8. RNN 14
8.1. Model Structure 14
8.2. Experimental setup 15
8.3. Results 16
9. Summary and Conclusion 17
9.1. Conclusion 17
9.2. Contribution 17
9.3. Future works 18
10. References 18

1. Introduction

All planets in our solar system orbit around the sun. Planets that orbit around other stars are called exoplanets. There's a lot of questions that motivate the study of exoplanets, such as how do we look for exoplanets, how can we find Earth-like planets in other solar systems, how many exoplanets exist, how diverse are exoplanets, what kind of physical character do exoplanets contain, etc. Thousands of exoplanets have been discovered within the past 20 years. Because exoplanets are hidden by the bright glare of the stars they orbit, they are very difficult to see directly with telescopes. NASA's Kepler Space Telescope was launched, and was mostly used to obtain these discoveries. Although in 2018, the Kepler mission ended, the Transiting Exoplanet Survey Satellite (TESS) took over to find new exoplanets.

2. Problem Statement

Since NASA's Kepler Space Telescope data is still publicly available, it's possible to make predictions about which exoplanets might be susceptible to life. The goal of this project is to develop neural networks to successfully model data for the detection of exoplanets. Using Keras as the deep learning framework, these models will be able to predict the existence of an exoplanet utilizing the flux (light intensity) readings. Previously, others have researched the same topic and uploaded kernels of their work on Kaggle. We decided to use neural networks such as multilayer perceptron (MLP), convolutional (CNN) and recurrent (RNN) as an aid to build our own models. In order to improve our models' performance to our best capability, we tried different preprocessing methods and focused on addressing overfitting problems, which allowed us to evaluate our models from several perspectives.

3. Description of Data

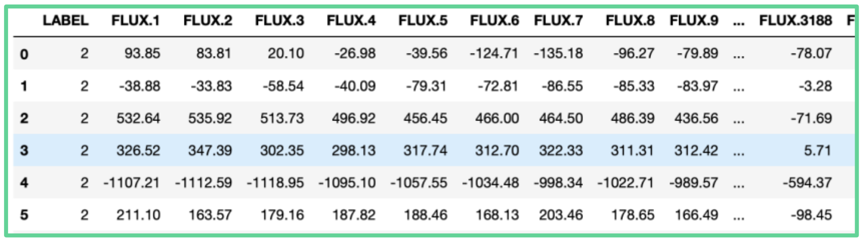
3.1. Source of Data

The dataset was provided by Kaggle, but originally derived from observations made by NASA Kepler Space Telescope.

The data describes the change in flux (light intensity) of several thousand stars. Each star has a binary label of "2" or "1". Label "2" indicates that the star is confirmed to have at least one exoplanet in orbit; some observations are in fact multi-planet systems.

3.2. Overview of Dataset

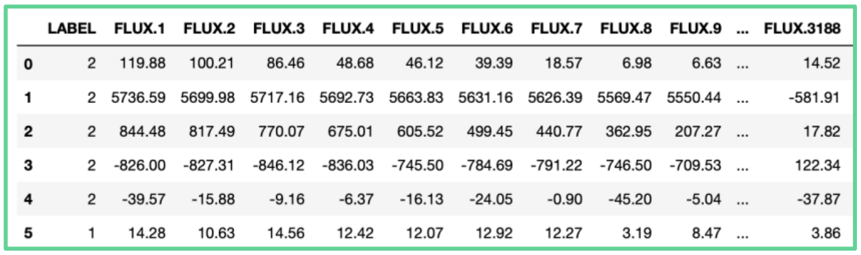
A brief description of dataset:



	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	...	FLUX.3188	F
0	2	93.85	83.81	20.10	-26.98	-39.56	-124.71	-135.18	-96.27	-79.89	...	-78.07	
1	2	-38.88	-33.83	-58.54	-40.09	-79.31	-72.81	-86.55	-85.33	-83.97	...	-3.28	
2	2	532.64	535.92	513.73	496.92	456.45	466.00	464.50	486.39	436.56	...	-71.69	
3	2	326.52	347.39	302.35	298.13	317.74	312.70	322.33	311.31	312.42	...	5.71	
4	2	-1107.21	-1112.59	-1118.95	-1095.10	-1057.55	-1034.48	-998.34	-1022.71	-989.57	...	-594.37	
5	2	211.10	163.57	179.16	187.82	188.46	168.13	203.46	178.65	166.49	...	-98.45	

Train set:

- 5087 observations
- 3198 features
    - Column 1- Label 1: exoplanet star; Label 2: non-exoplanet star
    - Column 2 - Flux values over time



	LABEL	FLUX.1	FLUX.2	FLUX.3	FLUX.4	FLUX.5	FLUX.6	FLUX.7	FLUX.8	FLUX.9	...	FLUX.3188
0	2	119.88	100.21	86.46	48.68	46.12	39.39	18.57	6.98	6.63	...	14.52
1	2	5736.59	5699.98	5717.16	5692.73	5663.83	5631.16	5626.39	5569.47	5550.44	...	-581.91
2	2	844.48	817.49	770.07	675.01	605.52	499.45	440.77	362.95	207.27	...	17.82
3	2	-826.00	-827.31	-846.12	-836.03	-745.50	-784.69	-791.22	-746.50	-709.53	...	122.34
4	2	-39.57	-15.88	-9.16	-6.37	-16.13	-24.05	-0.90	-45.20	-5.04	...	-37.87
5	1	14.28	10.63	14.56	12.42	12.07	12.92	12.27	3.19	8.47	...	3.86

Test set:

- 570 observations
- 3198 features (Same as Trainset)
- 5 confirmed exoplanet-stars
- 565 non-exoplanet-stars

4. Data Preprocessing

4.1. Data Scaling

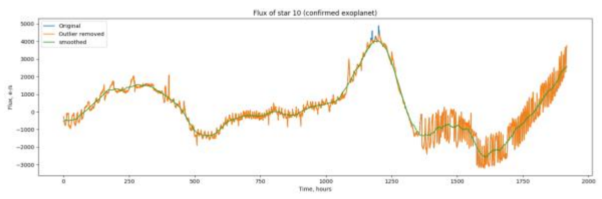
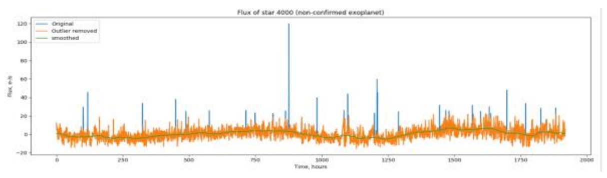
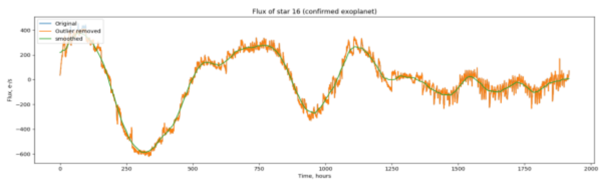
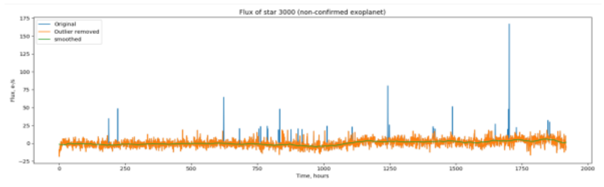
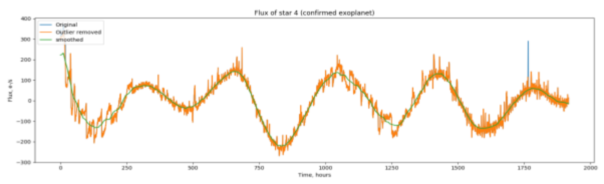
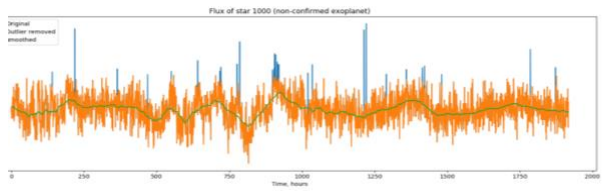
By visualizing the original data, we noticed that the shape and distributions of features were different. Also the spans of flux differ a lot, some have several thousand light intensity at maximum while some only have around thirty at max. We scaled the data by subtracting the mean value and dividing by the standard deviation.

4.2. Outlier Removing

Since we were looking for dips in flux recorded by the telescope, we removed the upper outliers.

4.3. Data Resampling

Our training data was extremely imbalanced, we have over five thousands negative samples but only twenty-nine positive samples. So we considered applying both oversampling and undersampling. We did research about how to find an exoplanet, one of the methods is called transit photometry that is based on the light intensity. There is a special relationship between the brightness of the planet and time due to the orbiting feature of exoplanets. The light changes periodically and the light intensity curve moves down and up in each period. With that in mind, we rotated the flux by time and then we got dummy exoplanets to increase the number of positive samples. We also compared the trend of the light intensity of exoplanet and that of non-exoplanet. We saw that the general curve for non-exoplanet is almost flat, especially for the first two. So with this theory, it is appropriate to get more positive samples by rotating the flux by time. Indeed, it improves the f1-score of our models a lot.

Exoplanet	**Non-Exoplanet**
	
	
	

4.4. Data Relabelling

The original dataset labels exoplanet as 2 and non-exoplanet as 1. We relabel the exoplanet to 1 and non-exoplanet to 0 so that it can be treated as a classification problem.

5. Math Background

5.1. Loss: binary__crossentropy

Since we were training a binary classifier, the target is either exoplanet or non-exoplanet. We used binary cross-entropy as our loss function. The concept behind binary cross-entropy is clear and straightforward. Here we start from looking to understand the role of entropy. Entropy is a measure of the uncertainty associated with a given distribution q(y). We can compute the entropy of a distribution, like q(y), using the formula below, where C is the number of classes:

$$H(q) = -\sum_{c=1}^{C} q(y_c) \cdot log(q(y_c))$$

Entropy

So, if we know the true distribution of a random variable, we can compute its entropy. If we don't know the true distribution, we can compute entropy by assuming our points follow this other distribution p(y). But we know they are actually coming from the true (unknown) distribution q(y).

$$H_p(q) = -\sum_{c=1}^{C} q(y_c) \cdot log(p(y_c))$$

Cross-Entropy

If we match p(y) and q(y) perfectly, the computed values for cross-entropy and entropy will match as well. However, this is unlikely to ever happen, cross-entropy will have a bigger value than the entropy computed on the true distribution. The Kullback-Leibler Divergence, or "KL Divergence" for short, is a measure of dissimilarity between two distributions:

$$D_{KL}(q||p) = H_p(q) - H(q) = \sum_{c=1}^{C} q(y_c) \cdot [log(q(y_c)) - log(p(y_c))]$$

KL Divergence

The closer p(y) gets to q(y), the lower the divergence and the cross-entropy will be. So our classifier should find the best possible p(y), which is the one that minimizes the cross-entropy. During the training, the classifier uses each of the N observations to compute the cross-entropy loss, fitting the distribution p(y). Since the probability of each observation is 1/N, cross-entropy is given by:

$$q(y_i) = \frac{1}{N} \Rightarrow H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} log(p(y_i))$$

Cross-Entropy —point by point

We need to compute the cross-entropy on top of the probabilities associated with the true class of each point. So we compute the average of all observations in both classes, positive and negative:

$$\begin{gathered} y_i = 1 \Rightarrow log(p(y_i)) \\ y_i = 0 \Rightarrow log(1 - p(y_i)) \end{gathered}$$

$$H_p(q) = -\frac{1}{(N_{pos} + N_{neg})}\left[\sum_{i=1}^{N_{pos}} log(p(y_i)) + \sum_{i=1}^{N_{neg}} log(1 - p(y_i))\right]$$

Binary Cross-Entropy — computed over positive and negative classes

We can take any observations, either from the positive or negative class, under the same formula. In a word, Binary cross-entropy measures how far away from the true value (which is either 0 or 1) the prediction is for each of the classes and then averages these class-wise errors to obtain the final loss.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

5.2. Evaluation: F1

Since this is a classification problem, we want to look at the results of classification reports to assess the performance of our models. F1 score is an appropriate indicator, especially for an imbalanced dataset. Here we have a look at the mathematical intuition behind.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Precision measures about how precise/accurate the model is out of those predicted positive, how many of them are actual positive. Precision is a good measure to determine when the costs of False Positive is high.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

So Recall actually calculates how many of the Actual Positives the model captures through labeling it as Positive (True Positive).

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

F1 which is a function of Precision and Recall, the formula is as follows. F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

$$\text{F1} = 2 \text{ x } \frac{Precision * Recall}{Precision + Recall}$$

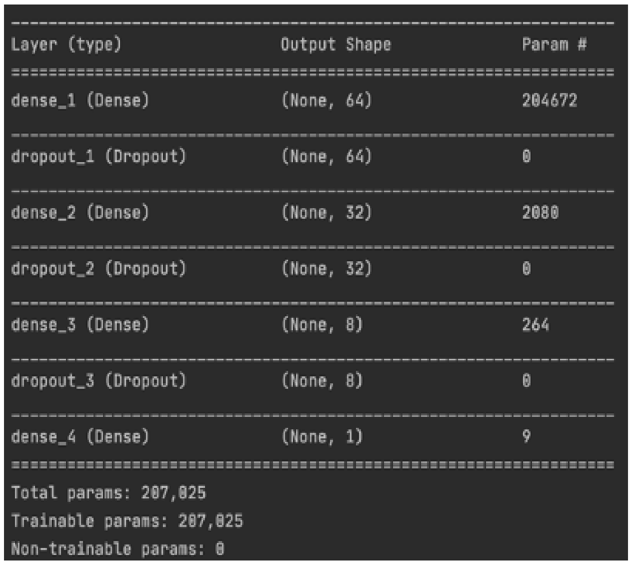
6. MLP

6.1. Model Structure

A base MLP model was used with 4 neuron hidden layers, a rectified linear activation function on hidden neurons, Dropout of 0.25 between each layer, and sigmoid logistic activation function on output neurons. A batch size of 3 with epochs of 15 and a batch size of 32 with epochs of 50 was used, with the training data. Normally, the training dataset is shuffled after each batch or each epoch, which can aid in fitting the training dataset on classification and regression problems. Shuffling was turned off, as it seemed to result in better performances. The model was compiled using the efficient ADAM or SGD optimization algorithm and binary cross entropy loss function.

6.2. Experimental Setup

Depending on the batch size, each experimental scenario ran between 5-7 minutes and the val_loss and accuracy score on the test set was recorded from the end of each run. To visualize the accuracy of the MLP model, a loss plot was conducted.

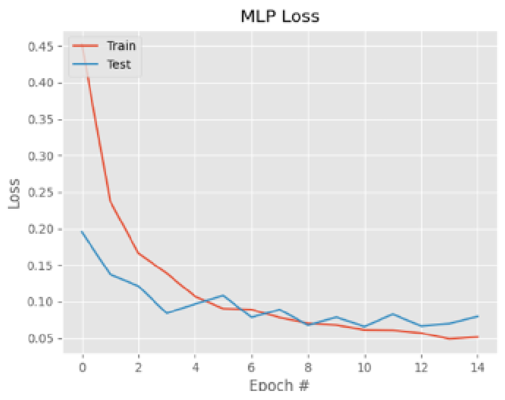
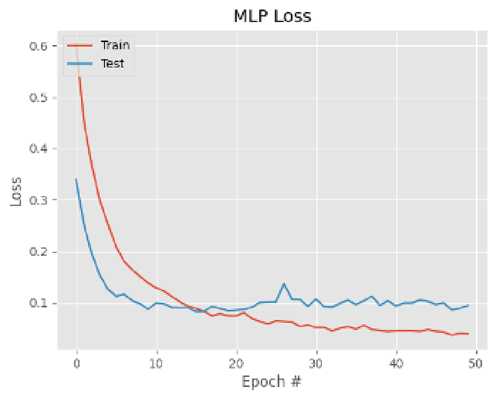
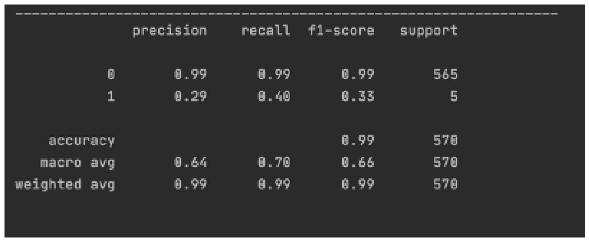
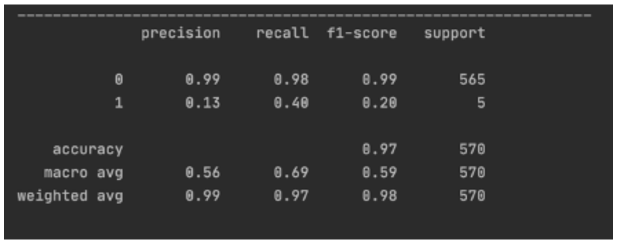


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                204672
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 8)                 264
_________________________________________________________________
dropout_3 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 9
=================================================================
Total params: 207,025
Trainable params: 207,025
Non-trainable params: 0
```

We investigated the training of a MLP with 4 hidden layers and 64, 32, 8, and 1 neuron(s) in its respective hidden layer, which produced 207,025 trainable parameters. There were no non-trainable parameters.

6.3. Results

Early stopping to avoid overfitting and to ensure the best model was displayed. No matter what the batch size or epochs was, class 0 (exoplanets) F1 score and accuracy would not change significantly. It ranged from 0.98 to 1.00 each time the code was run. We noticed as the batch size and epochs training decreased, the model performance improved for class 1 (non-exoplanets). Model 1 fit consisted of x_train and y_train sampling, batch size 3, and epochs 15. That model was compiled using the efficient ADAM optimization algorithm and binary cross entropy loss function. Model 2 fit consisted of x_train and y_train sampling, batch size 32 and epochs 50. That model was compiled using the efficient SGD optimization algorithm and binary cross entropy loss function.

Model 1 Loss Plot + Classification Report	**Model 2 Loss Plot + Classification Report**
	
 0 - exoplanet star 1 - non-exoplanet star	
Optimizer = ‘Adam’, Batch Size = 3, Epochs = 15	Optimizer = ‘SGD’, Batch Size = 32, Epochs = 50

Model 1:

```
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       565
           1       0.29      0.40      0.33         5

    accuracy                           0.99       570
   macro avg       0.64      0.70      0.66       570
weighted avg       0.99      0.99      0.99       570
```

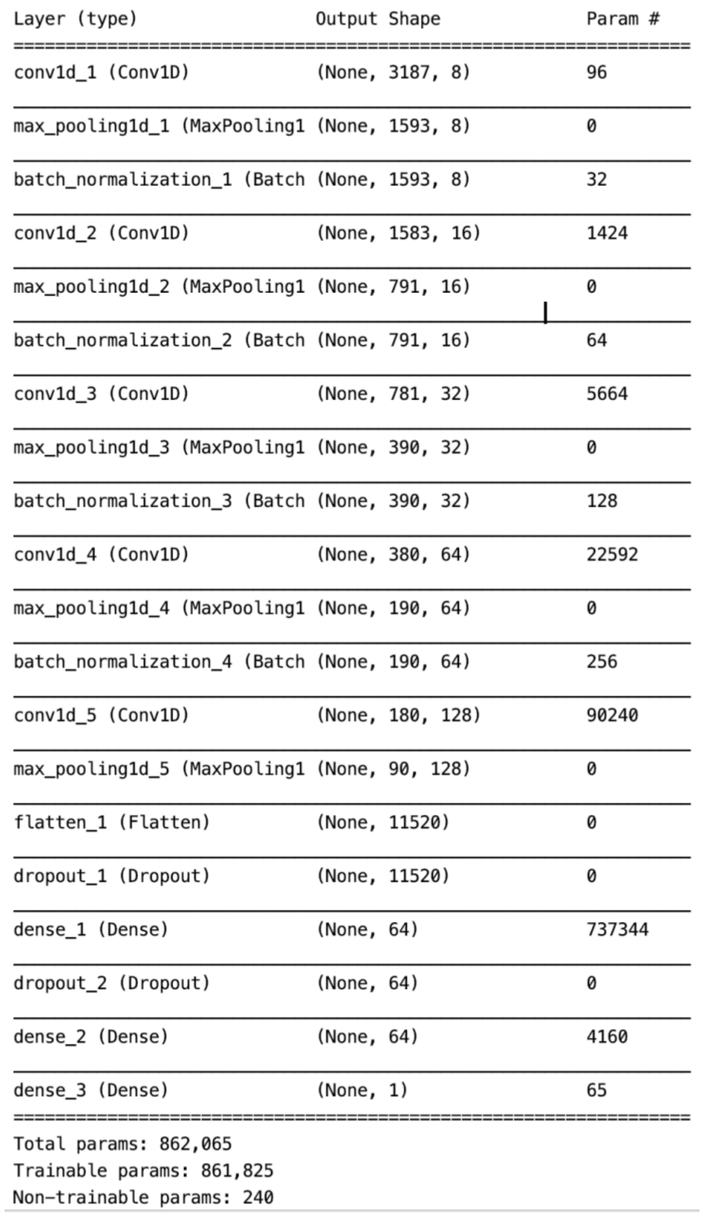
Model 2:

```
-----------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.99      0.98      0.99       565
           1       0.13      0.40      0.20         5

    accuracy                           0.97       570
   macro avg       0.56      0.69      0.59       570
weighted avg       0.99      0.97      0.98       570
```

Both MLP models did not perform well as the F1 score was very low and wouldn’t perform above 0.33. There were not many MLP models to compare these models too. It’s also important to note, the precision value 0.29 and 0.13 were fairly low as well. Maybe this chosen algorithm was not the best option for this problem. Perhaps CNN and RNN is a better fit to produce efficient models.

7. CNN

7.1. Model Structure



```
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 3187, 8)           96        
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 1593, 8)           0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 1593, 8)           32        
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 1583, 16)          1424      
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 791, 16)           0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 791, 16)           64        
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 781, 32)           5664      
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 390, 32)           0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 390, 32)           128       
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 380, 64)           22592     
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 190, 64)           0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 190, 64)           256       
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 180, 128)          90240     
_________________________________________________________________
max_pooling1d_5 (MaxPooling1 (None, 90, 128)           0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 11520)             0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 11520)             0         
_________________________________________________________________
dense_1 (Dense)              (None, 64)                737344    
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0         
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160      
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 65        
=================================================================
Total params: 862,065
Trainable params: 861,825
Non-trainable params: 240
```

The final CNN model chosen has 5 convolutional layers with channels from 8 to 128 and with max pooling layers and batch normalization layers connected each layer, and the kernel sizes are all 11. After model flatten, 2 dense layers both with 64 neuron(s) and dropout 0.5 and 0.25 in case overfitting was apploed. The activation function for the layers above are all 'relu'; and the final layer takes sigmoid activation function.

7.2. Experimental setup

Different convolutional layer amounts, kernel size, output amount, pooling method, dense layer amount, number of neurons in a dense layer was tested. Some of the results are listed below.

	Best Model (training all 0 samples)	**Kernel size = 13 for the first two convolutional layers**	**Using AvgPooling before flatten**	**2 more convolutional layers and more neurons in dense layers**
F1-score	**0.8**	**0.75**	**0.5**	**0.29**

At first, ‘Adam’ was used as the optimizer and the validation loss converges quickly but it fluctuates then. Early stopping was applied and model stops at 18 epochs with a bad f1-score. Early stopping included patience for the epochs and added model checkpoint as callbacks to save the best model. F1-score improves but not that big. Then ‘SGD’ was tried as the optimizer with more epochs(200) and the loss curve becomes smooth after it converges. Actually after around epoch 40, the loss becomes almost flat. Finally, the number of epochs 50 was chosen.

Since f1-score is an important reference of the goodness of the model, and in order to balance recall and precision, the threshold was changed to classify the model output to zero or one. Sometimes, lowering the threshold helps improve the recall of one. It came to realization that the test set is so imbalanced that it contains only five ones. The improvements brought by the threshold could be luck. Then the threshold was reset to be 0.5.

Each observation of a planet is a list of flux intensity by time. We wonder if measuring the overall trend and fluctuation of the light intensity is more efficient. We removed the noise by smoothing the data, which is usually done in time series problems. However, both precision and recall for positive samples become 0. We assumed that the data is ‘over-smoothed’, tried increasing the weights of the original data, but neither worked. Actually, we were using a time-series dataset but not solving a time-series problem; we are not forecasting or predicting the future light intensity. So considering the oscillation as features, we do not erase any noises; instead, we keep all the changes of light intensity by time.

7.3. Results

Preprocessing data plays a significant role in the performance of the model; especially for sampling in dealing with imbalanced data. We rotate the flux to generate dummy exoplanet data, the number of the ‘fake’ data generated matters. Since the gap of the amount of two categories is huge, it is not optimal to generate a lot of dummy exoplanets to fill the gap. Then a simple undersampling is helpful in balancing the data. In general, we drop similar data in undersampling; that is finding the nearest neighbors and drop a few. However, for each planet, there are over 3000 dimensions and it could be time-consuming to find the similarity. The first 3500 observations were taken from the dataset knowing that exoplanets(1s) are all arranged in the front rows. And it actually improves the model by balancing the data. Below are some of the approaches.

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       565
           1       0.75      0.60      0.67         5

    accuracy                           0.99       570
   macro avg       0.87      0.80      0.83       570
weighted avg       0.99      0.99      0.99       570
```

Model 1: *200 rotations (200*29 more 1's) | all non exoplanets (0's)*

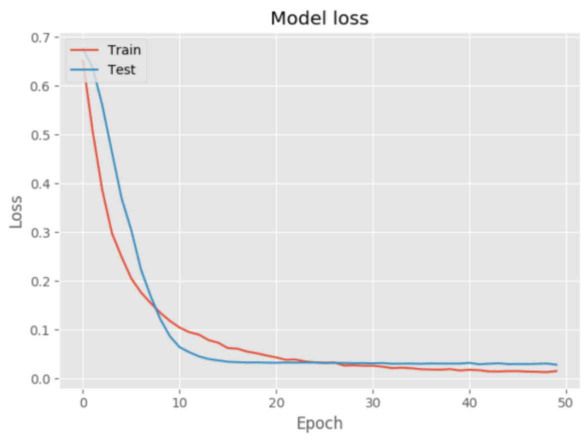
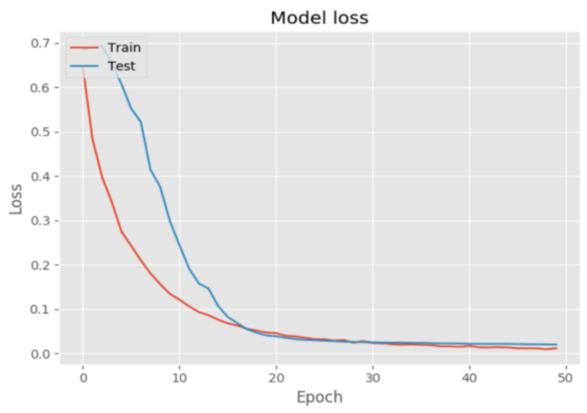
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       565
           1       0.80      0.80      0.80         5

    accuracy                           1.00       570
   macro avg       0.90      0.90      0.90       570
weighted avg       1.00      1.00      1.00       570
```

Model 2: *100 rotations (100*29 more 1's) | all non-exoplanets (0's)*

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       565
           1       0.83      1.00      0.91         5

    accuracy                           1.00       570
   macro avg       0.92      1.00      0.95       570
weighted avg       1.00      1.00      1.00       570
```

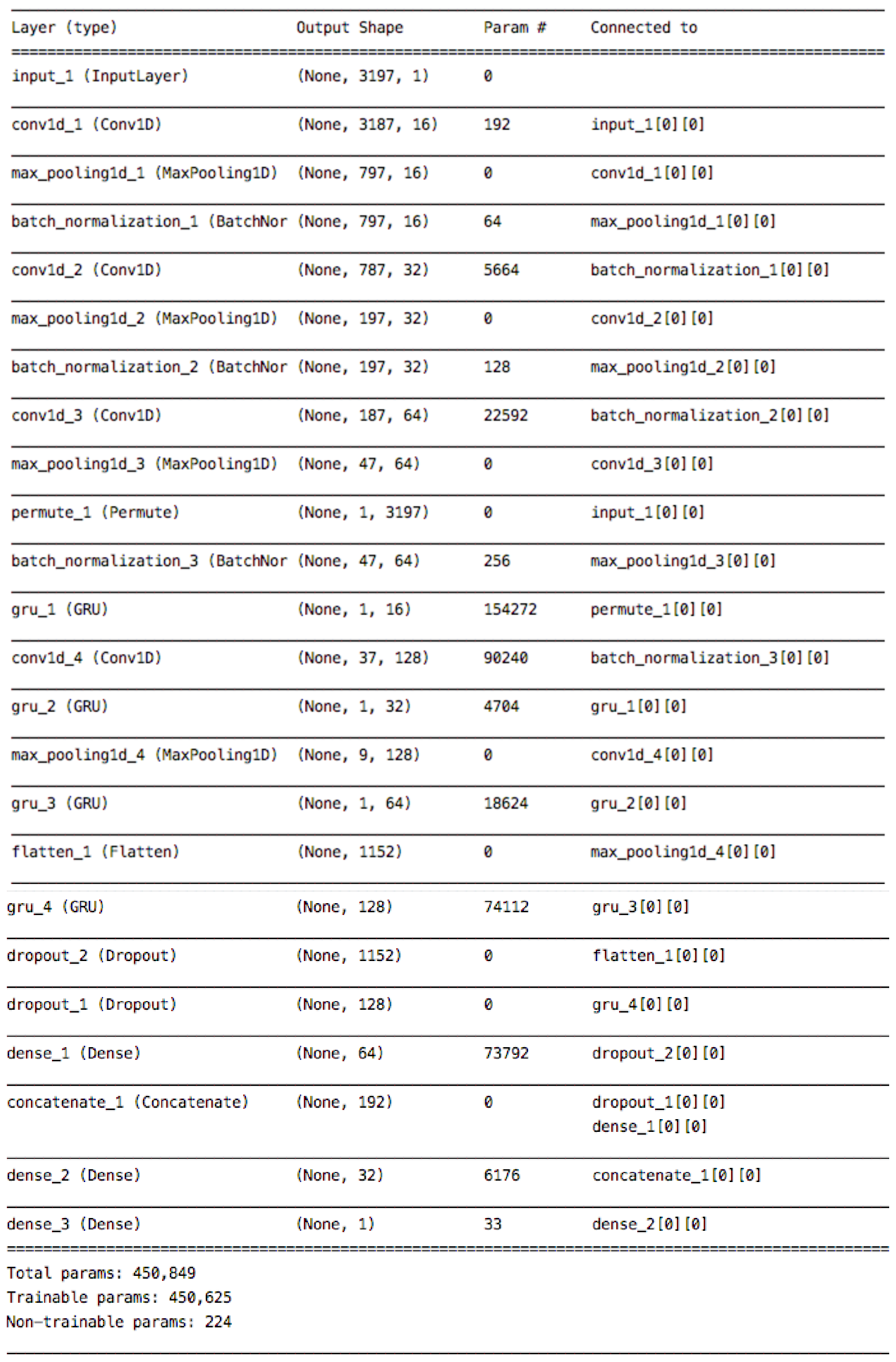
Model 3: *100 rotations (100*29 more 1's) | 3521 (1587 less) exoplanets (0's)*

Moreover, f1-score is not our unique consideration on model evaluation. Even though the validation loss of *Model 3* converges slower, it reaches the bottom level as *Model 2* does. With the obvious difference in f1-score and similar loss, we finally take *Model3.*

Model 2 Loss Plot	**Model 3 Loss Plot**
	

8. RNN

8.1. Model Structure



```
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            (None, 3197, 1)      0
__________________________________________________________________________________________________
conv1d_1 (Conv1D)               (None, 3187, 16)     192         input_1[0][0]
__________________________________________________________________________________________________
max_pooling1d_1 (MaxPooling1D)  (None, 797, 16)      0           conv1d_1[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 797, 16)      64          max_pooling1d_1[0][0]
__________________________________________________________________________________________________
conv1d_2 (Conv1D)               (None, 787, 32)      5664        batch_normalization_1[0][0]
__________________________________________________________________________________________________
max_pooling1d_2 (MaxPooling1D)  (None, 197, 32)      0           conv1d_2[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 197, 32)      128         max_pooling1d_2[0][0]
__________________________________________________________________________________________________
conv1d_3 (Conv1D)               (None, 187, 64)      22592       batch_normalization_2[0][0]
__________________________________________________________________________________________________
max_pooling1d_3 (MaxPooling1D)  (None, 47, 64)       0           conv1d_3[0][0]
__________________________________________________________________________________________________
permute_1 (Permute)             (None, 1, 3197)      0           input_1[0][0]
__________________________________________________________________________________________________
batch_normalization_3 (BatchNor (None, 47, 64)       256         max_pooling1d_3[0][0]
__________________________________________________________________________________________________
gru_1 (GRU)                     (None, 1, 16)        154272      permute_1[0][0]
__________________________________________________________________________________________________
conv1d_4 (Conv1D)               (None, 37, 128)      90240       batch_normalization_3[0][0]
__________________________________________________________________________________________________
gru_2 (GRU)                     (None, 1, 32)        4704        gru_1[0][0]
__________________________________________________________________________________________________
max_pooling1d_4 (MaxPooling1D)  (None, 9, 128)       0           conv1d_4[0][0]
__________________________________________________________________________________________________
gru_3 (GRU)                     (None, 1, 64)        18624       gru_2[0][0]
__________________________________________________________________________________________________
flatten_1 (Flatten)             (None, 1152)         0           max_pooling1d_4[0][0]
__________________________________________________________________________________________________
gru_4 (GRU)                     (None, 128)          74112       gru_3[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 1152)         0           flatten_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 128)          0           gru_4[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 64)           73792       dropout_2[0][0]
__________________________________________________________________________________________________
concatenate_1 (Concatenate)     (None, 192)          0           dropout_1[0][0]
                                                                 dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 32)           6176        concatenate_1[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 1)            33          dense_2[0][0]
==================================================================================================
Total params: 450,849
Trainable params: 450,625
Non-trainable params: 224
__________________________________________________________________________________________________
```

In the original model, LSTM was used in the recurrent layers. Both LSTM and GRU were applied and finally GRU was selected. The key difference between a GRU and an LSTM is that a GRU has two gates (reset and update gates) whereas an LSTM has three gates (namely input, output and forget gates).

FULL GRU Unit

$$\tilde{c}_t = \tanh(W_c[G_r * c_{t-1}, x_t] + b_c)$$

$$G_u = \sigma(W_u[c_{t-1}, x_t] + b_u)$$

$$G_r = \sigma(W_r[c_{t-1}, x_t] + b_r)$$

$$c_t = G_u * \tilde{c}_t + (1 - G_u) * c_{t-1}$$

$$a_t = c_t$$

LSTM Unit

$$\tilde{c}_t = \tanh(W_c[a_{t-1}, x_t] + b_c)$$

$$G_u = \sigma(W_u[a_{t-1}, x_t] + b_u)$$

$$G_f = \sigma(W_f[a_{t-1}, x_t] + b_f)$$

$$G_o = \sigma(W_o[a_{t-1}, x_t] + b_o)$$

$$c_t = G_u * \tilde{c}_t + G_f * c_{t-1}$$

$$a_t = G_o * tanh(c_t)$$

The GRU controls the flow of information like the LSTM unit, but without having to use a memory unit. It just exposes the full hidden content without any control. The performance of GRU is on par with LSTM, but computationally more efficient. As can be seen from the equations LSTMs have a separate update gate and forget gate. This clearly makes LSTMs more sophisticated but at the same time more complex as well.

8.2. Experimental setup

(1) Learning rate: 0.01

(2) Optimizer: SGD
The original model that was borrowed from was using Adam as the optimizer. When plotting the loss for the original model, the validation loss was very fluctuated. We guessed that Adam might have been too aggressive to make the validation loss decrease step by step. Since we already split the training dataset into small batches, mini batch gradient descent can be used to calculate

model error and update model coefficients. We then tried SGD and found the validation loss gets smoother even using SGD makes the training process converge slower than using Adam.

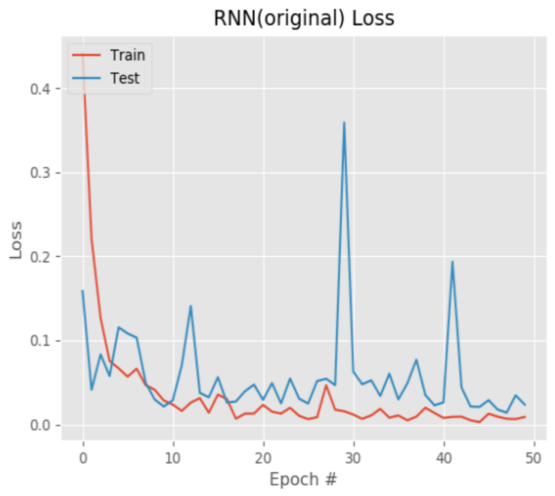
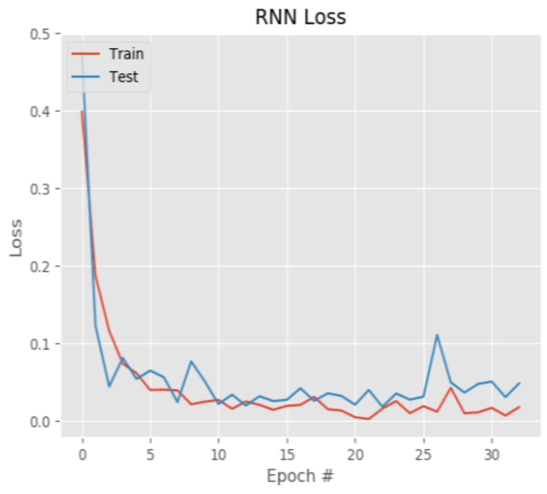
Adam is a widely used optimizer due to its power. But after a while we started noticing that despite superior training time, Adam in some areas does not converge to an optimal solution. Some research revealed that adaptive methods (such as Adam or Adadelta) do not generalize as well as SGD with momentum when tested on a diverse set of deep learning tasks.

(3) Batch generator: batch size = 32
Before modelling, we created a function called batch_generator. The batch size was set to 32, and the input of the function is x_train and y_train. This batch generator gave an equal number of positive and negative samples, and then we used np.roll() to randomly rotate those samples. We transferred the original data into batches which was used as the input of the model.

(4) Loss: binary_crossentropy

(5) Callbacks: Early stopping
Here we set the model stop training once the validation loss is no longer decreased within 10 epochs (monitor=val_loss, patience = 10).

8.3. Results

Original model	My model
	

Original model

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       565
         1.0       1.00      0.80      0.89         5

    accuracy                           1.00       570
   macro avg       1.00      0.90      0.94       570
weighted avg       1.00      1.00      1.00       570
```

My model

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       565
         1.0       0.67      0.80      0.73         5

    accuracy                           0.99       570
   macro avg       0.83      0.90      0.86       570
weighted avg       1.00      0.99      0.99       570
```

Compared to the results of the original model, the validation loss of our model is more smoothed, and both train loss and validation loss are showing a gradual downward trend. However, the precision value(0.67) and the F1 score(0.73) of our model are not as high as the original results.

9. Summary and Conclusion

9.1. Conclusion

Comparing these three models, we find that CNN performs best. Using CNN classifier, the precision value is 0.83, recall value is 1 and the F1 score for class 1 (non-exoplanet star) can reach to 0.91.

9.2. Contribution

First, we tried different ways to balance our data. Unlike typical oversampling and undersampling methods in sklearn, what we did was first generate negative samples (non-exoplanet star) to the half number of positive samples by rotating them, and then we only used almost half of the numbers of positive samples rather than all of them to train the models. We found this way performs better than random oversampling or SMOTE.

Second, we tried to adopt a smooth method on Kaggle, but we found the score getting lower. The smooth method was using uniform_filter1d to smooth features (flux) on each observation (star). The purpose was trying to move out noise from data. In general, the flux of exoplanets is more fluctuated than that of non-exoplanet. But when we plot some non-exoplanet samples before and after smoothing, we find that some flux shape has been changed. So we think it may not be appropriate to use the same smoothing method for all features.

Third, compared with other people's analysis on Kaggle, we use both loss value and F1 score as evaluation criteria. We all know that F1 score should be used as an evaluation when we deal with an imbalanced dataset. However, this dataset is so imbalanced that no matter what kinds of neural networks we use, the F1 score for positive samples can reach to almost 1. There are only 5 negative samples but 565 positive samples in the test set (hold out set). Therefore, we do not regard classification reports as the only criterion. Since we found that some of the results on the Kaggle have a very high F1, but they are actually overfitted. So on the other hand, we pay more attention to the loss value to prevent the model from overfitting

9.3. Future works

First, in order to further improve the smooth method, we believe that stars can be classified according to the distribution shape of their flux. Even the stars all belong to exoplanet stars, but the shape of flux may be different, some are sin or cosine shape, some are line shape and others are curves. So we'd better to classify those stars and adopt different smoothing methods to different sub-categories.

Second, because this data set is too unbalanced, in addition to the need to balance the data during training, a more scientific method, such as k-fold validation should be used when assessing the performance of models.

10. References

[1]https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a
[2]https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9
[3]https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c
[4]"Exoplanet Hunting in Deep Space". Kaggle. https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data. May 2020
[5]Brownlee, Jason. "How to Develop Multilayer Perceptron Models for Time Series". Machine Learning Mastery. https://machinelearningmastery.com/how-to-develop-multilayer-perceptron-models-for-time-series-forecasting/. May 2020
[6]Brownlee, Jason. "How to Stop Training Deep Neural Networks". https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/. May 2020
[7]"What is an Exoplanet" NASA Science, Space Place. https://spaceplace.nasa.gov/all-about-exoplanets/en/. May 2020