Udacity Machine Learning Nanodegree

Capstone Project: Exoplanet Hunting in Deep Space

Graciano Patino

November 2018

I. Definition

Project Overview

This project is about exoplanet hunting in deep space by using machine learning techniques on time series images from Nasa Kepler space telescope. The value of finding this type of planets is purely scientific (at this time). Finding an exoplanet could help NASA identify candidates of Earth-like candidate planets that perhaps in a not so-far future could be considered for deep space exploration.

Domain background

The domain for this project is the field of astronomy and astrophysics. In particular, the domain is about the fascinating topic of the search for new Earths, also called exoplanets. An exoplanet is a planet outside our Sun's solar system. The first evidence of an exoplanet was noted as early as 1917, but was not recognized as such. However, the first scientific detection of an exoplanet was in 1988, although it was not confirmed to be an exoplanet until later in 2012. The first confirmed detection occurred in 1992. As of 1 September 2018, there are 3,823 confirmed planets in 2,860 systems, with 632 systems having more than one planet. (Reference 1).

In the past few decades, scientists now have at their disposal more appropriate tools for detecting these planets. The Kepler space observatory provides images that can be analyzed on the search for exoplanets.

Problem Statement

The problem presented in this project was selected from the set of public datasets available at the Kaggle website (reference 2). Additional information is provided in the Github associated to this project (Reference 3).

The mission is to build a classification algorithm for identifying if a particular time series input includes an exoplanet or not. Basically, it is a binary classifier.

From the Overview tab in the mentioned kaggle dataset (above), this is high level description of the science involved on finding the exoplanets:

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

- Planets themselves do not emit light, but the stars that they orbit do. If said star is watched over several months or years, there may be a regular 'dimming' of the flux (the light intensity). This is evidence that there may be an orbiting body around the star; such a star could be considered to be a 'candidate' system. Further study of our candidate system, for example by a satellite that captures light at a different wavelength, could solidify the belief that the candidate can in fact be 'confirmed'.

Solution statement

The mission as stated in the Github (Reference 3) is to build a classification algorithm for identifying if a particular time series input includes an exoplanet or not. It also mentions that a number of methods were tested: 1-D CNN in Torch7, XGBoost in R and PCA in Python. However, none of these methods provided strong results according to source. Basically, no exoplanets were detected in the XGBoost and PCA cases. There is no mentioning of the result for the 1-D CNN. Arguably it was probably as bad as the other two.

It is important to mention that while reference 3 mentions that they were not successful on using 1-D CNN. The paper in reference 4, however, points to a very successful use of 1-D CNN with a network architecture than according to the authors is able to rank individual candidates 98.8% of the time. The models implemented in that paper were done in TensorFlow.

For this project, deep learning algorithms are evaluated. The steps used for the solution are as follows:

1) Initially I would evaluate 1-D CNN using Keras on TensorFlow instead of Torch7.
2) Based on reference paper (reference 4), I would try adding different number of layers and filters in combination with other CNN parameters. Details would be included in project report.
3) The output of the CNNs would be the input to one or more dense layers.
4) Grid search will be used for configuring the hyperparameters for the solution model(s).
5) Performance of each model to be measured as per evaluation metrics section.
6) Per Kaggle source the test set is confirmed to have 5 exoplanets. This will also be useful on checking performance of algorithms. If an algorithm is unable to identify (at least) one exoplanet on then testing set, then model might not be good.

Metrics

For assessing the models performance, the following metrics will be used (reference 5):

- Receiver Operating Characteristic (ROC) metric to evaluate classifier output quality.
    - ROC Area Under Curve (AUC) is one of the most widely used metrics for evaluation. It is used for binary classification problems. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.
    - An important feature of AUC is that it is independent of the choice of the classification threshold.
    - In particular, we would attempt to maximize AUC from prediction scores.
- Precision: the fraction of signals classified as planets that are true planets (also known as reliability, per reference 4).
- Recall: the fraction of true planets that are classified as planets (per reference 4).

- Precision and Recall, both, depend on the choice of the classification threshold.
- Confusion Matrix as the name suggests gives us a matrix as output and describes the complete performance of the model.
- F1 Score is used to measure a test's accuracy. F1 Score is the Harmonic Mean between precision and recall. The range for F1 Score is [0, 1]. It tells you how precise your classifier is (how many instances it classifies correctly), as well as how robust it is (it does not miss a significant number of instances).
    - The greater the F1 Score, the better is the performance of our model.
    - F1 Score is computed using the Precision and Recall metrics.
- Given that the dataset for this problem is very unbalanced, less than 0.1% of samples are exoplanets, the accuracy metric isn't appropriate for evaluating performance on the model.

II. Analysis

Data Exploration

This project is possible due to data images provided by the NASA Kepler space telescope project. They provided clean data and posted the problem a sample set in Kaggle (Reference 2). Basically, the data provided is in the form of time series of flux (light intensity) values taken for a number of stars. For each star the time series contains history of the change in flux. Each time a change of flux is detected this opens the possibility that a new exoplanet candidate could be identified (classified).

As described in Kaggle for this proposal (Reference 2). The following training and testing sets are provided in two comma separated values file (CSV). The data is clean according to the Kaggle source.

Training set (File: exoTrain.csv):

- 5087 rows or observations.
- 3198 columns or features.
- Column 1 is the label vector. Columns 2 - 3198 are the flux values over time.
- 37 confirmed exoplanet-stars and 5050 non-exoplanet-stars.

Testing set (File: exoTest.csv):

- 570 rows or observations.
- 3198 columns or features.
- Column 1 is the label vector. Columns 2 - 3198 are the flux values over time.
- 5 confirmed exoplanet-stars and 565 non-exoplanet-stars.

It is noted in Github, that the datasets are not normalized. Therefore, normalization would have occur prior to training the model.

Observations from the data exploration:

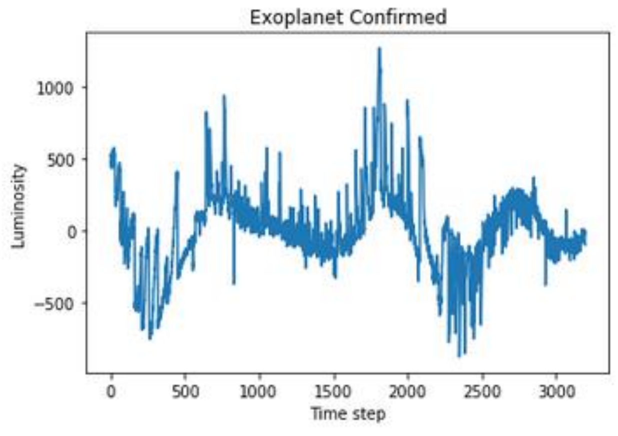
- The datasets are heavily unbalanced. There are only 37 exoplanets out of 5087 observations in the training dataset.

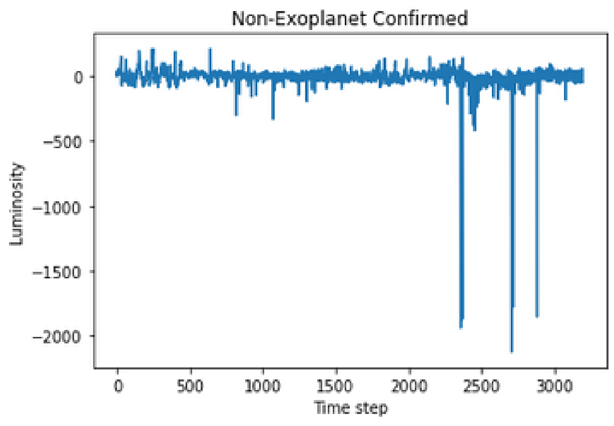
- There are only 5 confirmed exoplanets out of 570 observations in the testing dataset.
- The exoplanets are marked with a 2. Need to be converted to 1’s.
- The non-exoplanets are marked with a 1. Need to be converted to 0’s.
- By simple exploration of the observations, it was clear that each observation row contained a very wide range of values. It was necessary to normalize every observation, except the label, by applying Standard Scaler from scikit-learn such that each time series would have mean zero and variance one.

The datasets were checked in such a way to find out that there were no NULL values in any entry. This was performed using the “isnull” routine from Pandas (on data frames). There were no NULLs identified so the data was clean on this respect.

Exploratory Visualization

In the pictures below, there are two examples of the time series contained in each row. There is one example of a confirmed exoplanet. There is another example of a confirmed non-exoplanet.





Each row corresponds to one observation of the flux time series. It can be clearly observed that each row has a wide range of values. It is important to consider that each row has more than 5,000 samples. Therefore, this is just a little window of all the observation. The range values could be actually much wider. This clearly shows the reason why we need to normalize the data. For this project, we are scaling the data to zero mean and variance one.

The code used to produce this graphs is presented in the file with name part of this project submission: exoplanet_search_data_exploration.ipynb

Algorithms and Techniques

Deep learning has been selected as the machine learning algorithm for solving this problem. Per reference 4, deep learning is a type of representation learning that uses computational layers to build increasingly complex features that are useful for classification problems.

In particular, we will be using a 1-dimension convolutional network (1-D CNN). CNNs exploit spatial structure by learning local features that are detected across the entire input. (Reference 4)

A CNN typically consists of convolutional layers and pooling layers. The output of the convolutional layers is fed to one (or more) dense neural networks. For a classification problem, the output layer is normalized so that each filter/neuron's value lies in (0, 1) and represents the probability of a specific output class.

Configuration of deep learning networks could become a complex task as the number of layers for CNNs and DNNs increase. The number of hyperparameters could become large making the task of finding the hyperparameters value not trivial. The configuration of each parameter would be affected by the dataset characteristics. In other words, there are no recipes to follow. To help on handling the task of configuring the model hyperparameters, we could use a technique called grid search. This technique allows to do a parametric search for the parameter candidate values for configuring the neural network models.

As mentioned in the problem statement, each row corresponds to a time series of the change in flux (light intensity). The input to the network will be a 1-dimensional vector (row) with 3,197 measurements of change in flux from the datasets, training and testing respectively.

For training the network model, we will use the golden rule of not using the testing data for training and validating the model. In order to accomplish this, we would use the scikit-learn function name train_test_split that will allow us to separate the training set into two new sets: new training set and validation set. Conveniently, this function allows you stratify the data in such a way that you can have a similar proportion of exoplanets in the new training and validation sets.

Benchmark

As described in the solution statement section at least 2 of the models attempted (XGBoost and PCA) were unsuccessful on classifying any exoplanets. Arguably, practically any model that successfully classifies at least one exoplanet would already be more successful.

As mentioned also in the solution statement section, the paper mentioned in reference 4 points to successful use 1-D CNN with a network architecture than according to the authors is able to rank individual candidates 98.8% of the time. Therefore their model represents a good example of a good implementation on 1-D CNN. Their solution was able to achieve ROC-AUC over 0.97. (A description of this metric is expanded in the evaluation metrics section.)

It is important to note that in the implementation of the model described in reference 4, the researchers had vast amounts of computing resources as they were able to simulate thousands of hyperparameters that allowed them to come up with the optimal parameters for their model.

For this project, we will attempt to provide a classifier model with an AUC score between 0.7 and 0.9. Per reference 9, depending on AUC score we have the following classification: Fail (0.5-0.6), Poor (0.6-0.7), Fair (0.7-0.8), Good (0.8-0.9), and Excellent (0.9-1.0). Therefore, we will attempt to get a result from fair to good. Targeting excellent would very likely require much more time.

III. Methodology

Data Preprocessing

The following steps were applied to the each one of the original datasets, training and testing, as provided in the Kaggle website (Reference 2):

1) Load data using Pandas.
2) Separate the time series (X) and labels (y) using Pandas; and convert to Numpy arrays.
3) Subtract one from each label as we want to have 0 for non-exoplanet and 1 for exoplanet.
4) Since range of values in each series (row) is very wide, we use scikit-learn StandardScaler to make each time series have zero mean and variance one.
    a. Since sets are big but not extremely large we can make this by transposing the time series matrix.
    b. Once matrix is transposed on the time series, apply the StandardScaler.
    c. Then transpose the matrix again to return to the original shape.

Note that time series values have to be normalized. As described in Kaggle each row has a wide range of values. This would make the job of the classifier much harder. By limiting the range of the values, we improve our chances of making a good classification (meaning with a good F1 score; closer to 1 is better).

Since original sets include only a training set and a testing set, we want to split the provided training set in two after applying the StandardScaler. This would end with a new (smaller) training set and a validation set that will be used for the validation process. For doing this, we apply the following steps:

1) From scikit-learn model_selection, we import train_test _split.
2) This function allows you stratify the data in such a way that you can have a similar portion of exoplanets in the new training and validation sets. Since we have an extremely low number of exoplanet samples, we don’t want to end up with a training or a validation set that includes all the exoplanets while the other set includes none.

In order to try to improve performance of the classifier, we can attempt to augment the data. For this purpose, the following Numpy routines for array manipulation were considered:

1) fliplr (reversing the order of the time series). This is the routine that was implemented in the project.
2) roll: Roll array elements along a given axis. For instance, you can shift the series by 200 samples. This would be equivalent to the augmentations performed on images like shifting the image a few steps such that classifier gets another view of the same image. This other technique was considered in case more augmentations was desired.

Note that this order reversal was applied to the confirmed exoplanets of the training dataset (after the train_test_split process mentioned above). This allowed to create new rows on confirmed exoplanets. These new rows were added to the training set such that the model would see more confirmed exoplanets. This was done as the datasets are heavily unbalanced in favor to non-exoplanets.

Implementation

For the implementation of the 1-D CNN, grid search was used for identifying candidate values for the hyperparameters of the model. There were two rounds of the grid search method for finding the candidate values for the network hyperparameters. The final solution was based primarily from the second grid search round (GS2). However, some parameters from grid search round 1 (GS1) were useful when a few parameters from GS2 did not work as expected.

The picture below depicts the network that was implemented. It contains 8 CNN layers, where the first layer is the input layer. The output of the CNNs is fed to two DNNs. The final output of the last DNN has a single output with activation sigmoid such that the output corresponds to the probability the corresponding time series is an exoplanet. (See figure 1 below.)

The grid search implementation used to finding the hyperparameters can be found in the folder named “Grid Search v2” with the code submitted for this project. Reference 7 on “How to Grid Search Hyperparameters for Deep Learning Models in Python with Keras” offers a good reference on the process involved and provides good examples on the topic.

It is important to mention the following:

1) Grid Search v1 (GS1): The searched parameters were for a smaller network (3 CNN layers, 2 DNN layers).
2) Grid Search v2 (GS2): The searched parameters were for a larger network (8 CNN layers, 2 DNN layers).
3) Some parameters as discussed in reference 4 were implemented at the end. This was the case for the Adam optimizer.

Grid Search results:

1) Parameter(s) searched: batch_size and number of epochs
    a. The batch size in iterative gradient descent is the number of patterns shown to the network before the weights are updated. It is also an optimization in the training

of the network, defining how many patterns to read at a time and keep in memory. (Reference 7)

b. The number of epochs is the number of times that the entire training dataset is shown to the network during training. (Reference 7)

c. Filename in Grid Search v2 directory: exoplanet_Nov04_GS_BatchSize_Epochs.ipynb

d. Result: batch_size=80 and epochs=10.

e. From GS1, we have one result of batch_size=80 and epochs=50.

f. Comments: As it turns out, batch_size=80 appeared to work fine. However, number of epochs (10) turned out to be too small as model had not converged yet (this would be observed in the results section).

g. Number of epochs ended being somewhere in the middle. Like epochs=30.

h. Per reference 7, we have to be careful as CNNs might be sensible to number of epochs.

2) Parameter searched: dropout regularization.

a. Tuning the dropout rate for regularization in an effort to limit overfitting and improve the model's ability to generalize. (Reference 7)

b. Filename in Grid Search v2 directory: exoplanet_Nov04_GS_dropout.ipynb

c. Result: 0.2 for GS2 (large network); 0.3 for GS1 (small network).

d. Comment: As it would be seen in results section. Dropout of 0.2 worked fine.

e. It is also important to note that the paper in reference 4 (5.2. Best Model Configuration section), mentions that for their best model the researchers did not use dropout regularization.

3) Parameter searched: optimizers.

a. The choice of optimization algorithm for your deep learning model can mean the difference between good results in minutes, hours, and days. (Reference 7)

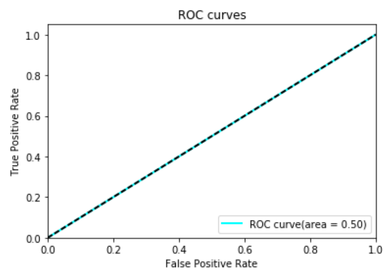
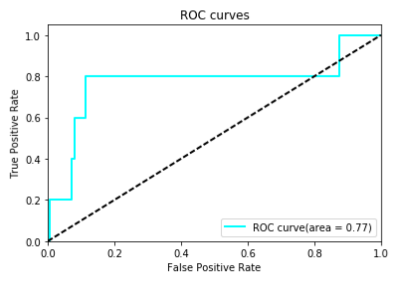
b. Filename in Grid Search v2 directory: exoplanet_Nov04_GS_optimizers.ipynb

c. Result: Per GS2, optimization Stochastic Gradient Decent (SGD) was the choice. However, per GS1, the option was in Adam.

d. It is also important to reference 4, the selected optimizer was also Adam (learning rate=1e-5, beta_1=0.9, beta_2=0.99). (Reference 8)

e. Per picture below, Adam was showing better results than SGD.

f. Comment:

SGD optimizer (AUC=0.5)	Adam optimizer (AUC=0.77)
 `Score: 0.500`	 `Score: 0.770`

4) Parameter searched: number of filters/neurons.
    a. The number of neurons in a layer is an important parameter to tune. Generally the number of neurons in a layer controls the representational capacity of the network, at least at that point in the topology. (Reference 7)
    b. Filename in Grid Search v2 directory: exoplanet_Nov06_GS_filters.ipynb
    c. Result: 4 for GS2 (large network); 12 for GS1 (small network).
    d. Please note that the number of filters is applied as a factor to specify the number of filters in a particular layer in the model. (See figure below.)
    e. Comment: As it would be seen in results section. Filter=4 works well.
5) Parameter searched: activation function in hidden layers.
    a. The activation function controls the non-linearity of individual neurons and when to fire. Generally, the rectifier activation function (ReLU) is the most popular, but it used to be the "sigmoid" and the "tanh" functions and these functions may still be more suitable for different problems. (Reference 7)
    b. Filename in Grid Search v2 directory: exoplanet_Nov06_GS_hidden_act_func.ipynb
    c. Result: The selected parameter was ReLU after testing the parameter in a network simulation.
    d. Comment: The result from GS2 was softmax. However, it produced an AUC=0.5. While activation ReLU produced AUC=0.78 and AUC=0.80 in two different runs.
    e. The derivative of the linear rectification function does not diminish as variable approaches +infinity, which helps avoid this "vanishing gradient problem" and often results in faster training and better performance.
6) Parameter searched: network weight initialization.
    a. This parameter is used to initialize the value of the weights in the network as different values might have an impact on how fast the network converges.
    b. Filename in Grid Search v2 directory: exoplanet_Nov06_GS_net_weight_init.ipynb
    c. Result: Uniform.
    d. Comment: Uniform was the network weight initialization chosen for all the model simulation runs after GS2.

Other parameters from the model:

1) MaxPool1D: This is used for maximum pooling for temporal data. (Keras)
2) BatchNormalization: Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1. (Keras)
3) For the model compilation, initially for a short while used "accuracy" as the metric used for judging the performance of the model with the validation data. Unfortunately, accuracy does not work well as metric for heavily unbalanced.
    a. After confirming that models run with "accuracy" metric were not performing consistently. Picked mean squared error (MSE) as the metric.
    b. MSE was picked as it is easier to compute a gradient. That is not the case to mean absolute error (MAE) which was considered for a short while.

    c. MSE pays more attention to big errors as we compute the square of the error.
4) The history of every run was tracked such to evaluate the learning process in a graph when comparing the training and validation metrics for the number of epochs run.
5) Callbacks were used such to keep the network parameters for the best runs during the model fitting process.
6) The kernel_size was adjusted from comparing runs during GS1 timeframe. Size of 6 was giving the best results at the time.
7) Flatten parameter was applied between output of convolutional layers and the input to the dense layers.

The final model is shown in the results section of this project report.

Refinement

The most important improvement out of a refinement came from removing the dropout regularization as mentioned in the previous section. The paper in reference 4 mentions that their best model the researchers did not use dropout regularization. While using Dropout(0.2) was working fine for the most part. There was a great improvement when it was completely removed from the network.

Model	AUC	Number of epochs
Best model with Dropout(0.2)	0.77	30
Best model w/o dropout	0.879	30

Note: The final solution will be presented in the results section.

As seen in the table above, removing dropout regularization had a very positive effect. It increased the AUC by above 14%. The model went from fair to good according to the AUC scale previously discussed. The boundary for very good is 90%.

A number of other refinements for the model were mentioned in the implementation section. For clarity purposes, the main refinements are listed below:

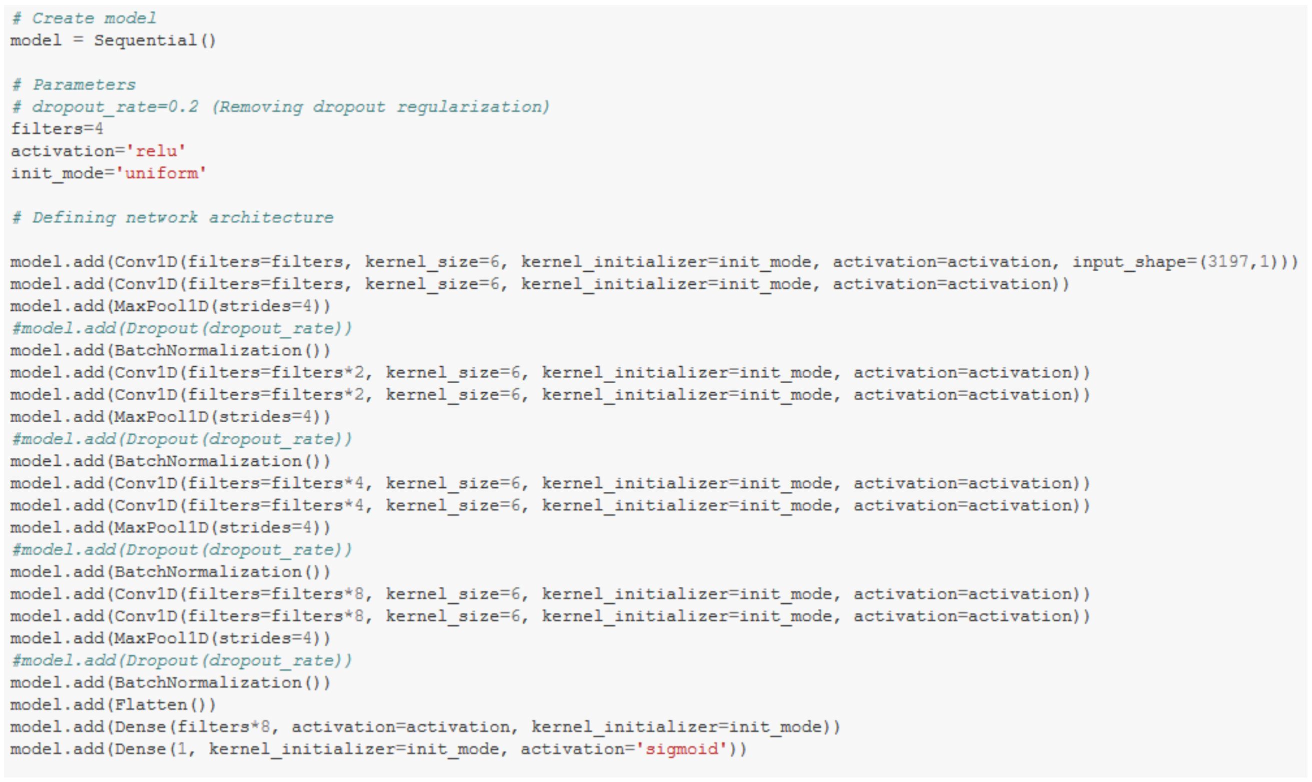
1) Removed dropout regularization as discussed earlier in this section.
2) Changed the optimizer from SGD to Adam, the difference was noticed right away. Results using Adam were better as noticed in the implementation section. AUC went from 0.5 to 0.77. Both optimizers had a learning rate of 1e-5.
3) Changed the compile metric from “accuracy” to “mse”. Inconsistent results with “accuracy”.
    a. Sometimes stumbling with what appeared to be a good model. AUC was over 0.8. However, the probabilities out of the sigmoid were very low (lower than 0.5).
    b. Also, the history graph showed that while the training accuracy improved as the number of epochs increased. The validation accuracy would be almost flat. Already high with very little room of improvement. Perhaps a form of overfitting was at play.
    c. After changing the metric to “mse” both the training and validation mse showed signs that there was learning as the number of epochs increase (until convergence).

4) Noticed that simulations were also sensitive to the number of epochs. Once the simulation history showed signs of convergence, you would notice there are slight changes on results after reaching convergence. Running many epochs especially after convergence have been reached would start impacting the probabilities out of the sigmoid function. For example, the same model with 30 epochs would be able to detect 4 true exoplanets correctly using threshold of 0.5. However, if you went to 50 epochs, the same model with same threshold would only detect 1 true exoplanet correctly and on top of that it would have an AUC slightly higher.
5) While the grid search came up with a softmax activation function for the hidden layers, it practice this parameter had poor results. Therefore it was changed immediately to ReLU which is very well know and it was also recommended in reference 4.
6) For the simulation, it was attempted to fix the random seed for reproducibility. The parameter is implemented in every single run.
7) The loss function for the compilation process was selected to be binary cross entropy as this is the typical for a binary classification project.

IV. Results

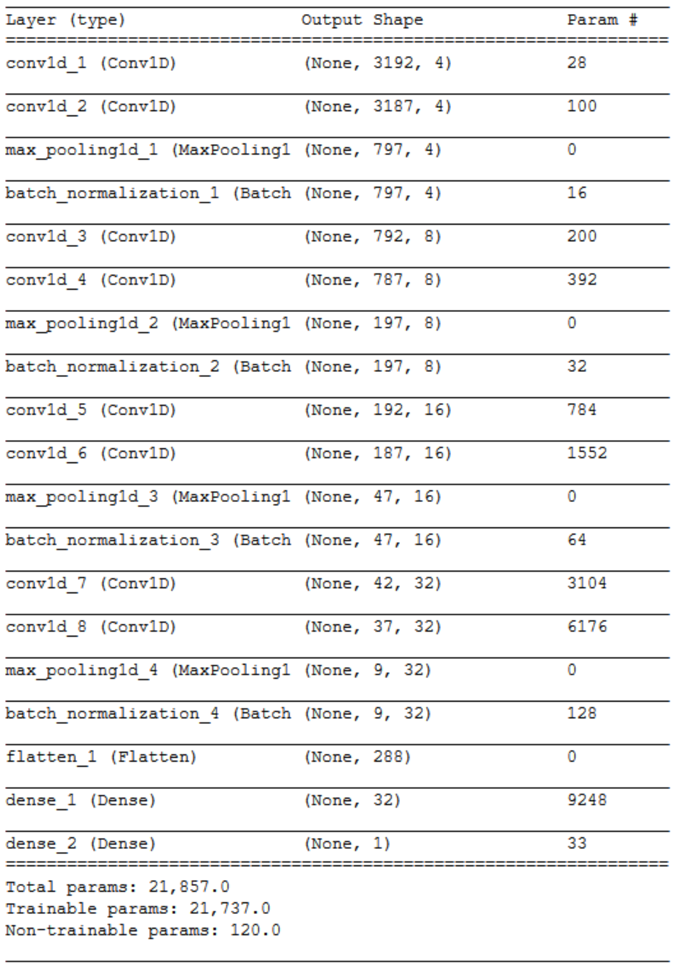
Model Evaluation and Validation

In this section, the final model and its results are presented. The final model is shown below.



```
# Create model
model = Sequential()

# Parameters
# dropout_rate=0.2 (Removing dropout regularization)
filters=4
activation='relu'
init_mode='uniform'

# Defining network architecture

model.add(Conv1D(filters=filters, kernel_size=6, kernel_initializer=init_mode, activation=activation, input_shape=(3197,1)))
model.add(Conv1D(filters=filters, kernel_size=6, kernel_initializer=init_mode, activation=activation))
model.add(MaxPool1D(strides=4))
#model.add(Dropout(dropout_rate))
model.add(BatchNormalization())
model.add(Conv1D(filters=filters*2, kernel_size=6, kernel_initializer=init_mode, activation=activation))
model.add(Conv1D(filters=filters*2, kernel_size=6, kernel_initializer=init_mode, activation=activation))
model.add(MaxPool1D(strides=4))
#model.add(Dropout(dropout_rate))
model.add(BatchNormalization())
model.add(Conv1D(filters=filters*4, kernel_size=6, kernel_initializer=init_mode, activation=activation))
model.add(Conv1D(filters=filters*4, kernel_size=6, kernel_initializer=init_mode, activation=activation))
model.add(MaxPool1D(strides=4))
#model.add(Dropout(dropout_rate))
model.add(BatchNormalization())
model.add(Conv1D(filters=filters*8, kernel_size=6, kernel_initializer=init_mode, activation=activation))
model.add(Conv1D(filters=filters*8, kernel_size=6, kernel_initializer=init_mode, activation=activation))
model.add(MaxPool1D(strides=4))
#model.add(Dropout(dropout_rate))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(filters*8, activation=activation, kernel_initializer=init_mode))
model.add(Dense(1, kernel_initializer=init_mode, activation='sigmoid'))
```

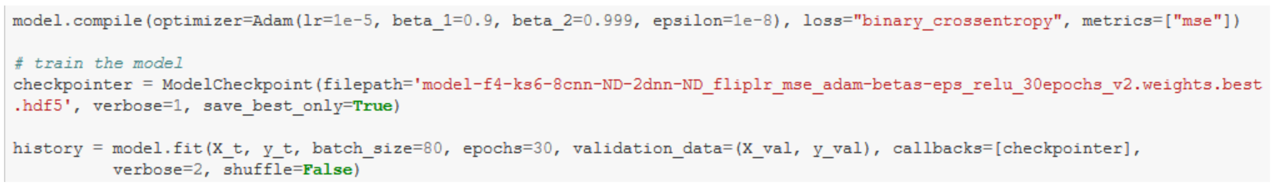
The parameters as it can be observed follow the refinements mentioned in the section with the same name (above). The model summary can be observed in Figure 2 below. As it can be observed this is the model without the dropout regularization. Interestingly enough the number of total parameters is not too large (as compared to the model in reference 4). This helped for running the models faster.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv1d_1 (Conv1D)            (None, 3192, 4)           28        
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 3187, 4)           100       
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 797, 4)            0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 797, 4)            16        
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 792, 8)            200       
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 787, 8)            392       
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 197, 8)            0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 197, 8)            32        
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 192, 16)           784       
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 187, 16)           1552      
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 47, 16)            0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 47, 16)            64        
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 42, 32)            3104      
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 37, 32)            6176      
_________________________________________________________________
max_pooling1d_4 (MaxPooling1 (None, 9, 32)             0         
_________________________________________________________________
batch_normalization_4 (Batch (None, 9, 32)             128       
_________________________________________________________________
flatten_1 (Flatten)          (None, 288)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 32)                9248      
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 33        
=================================================================
Total params: 21,857.0
Trainable params: 21,737.0
Non-trainable params: 120.0
_________________________________________________________________
```

Figure 2. Final model summary

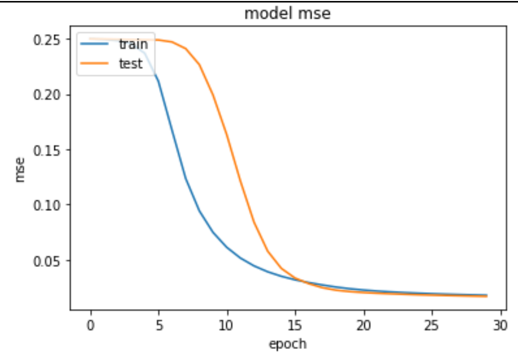
The model compilation and fitting parameters were as follows:

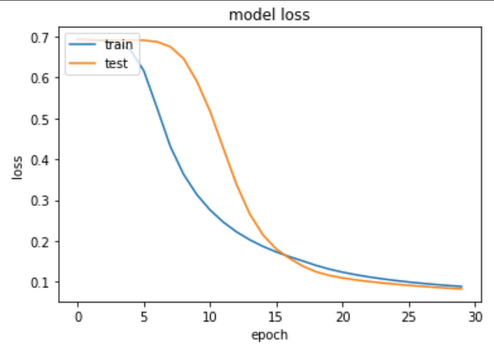


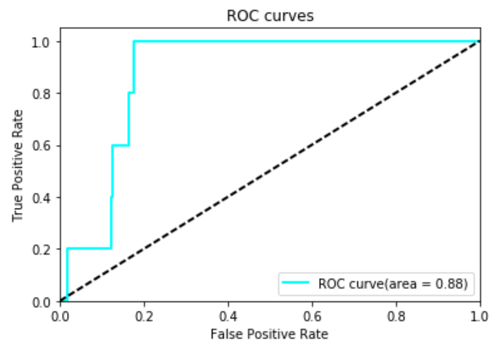
```
model.compile(optimizer=Adam(lr=1e-5, beta_1=0.9, beta_2=0.999, epsilon=1e-8), loss="binary_crossentropy", metrics=["mse"])

# train the model
checkpointer = ModelCheckpoint(filepath='model-f4-ks6-8cnn-ND-2dnn-ND_fliplr_mse_adam-betas-eps_relu_30epochs_v2.weights.best
.hdf5', verbose=1, save_best_only=True)

history = model.fit(X_t, y_t, batch_size=80, epochs=30, validation_data=(X_val, y_val), callbacks=[checkpointer],
          verbose=2, shuffle=False)
```

1) Optimizer is Adam is described in previous sections.
2) The selected model is "mae" as previously discussed.
3) Batch_size=80 and epochs=30. The number of epochs was adjusted after checking convergence via history graph. The graphs are shown below.

Graphs from mode on MSE, loss and AUC for the best model:







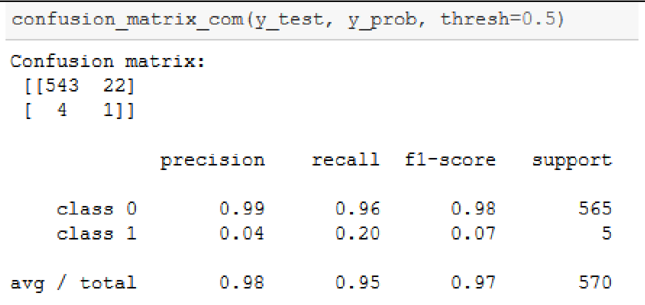
```
Score: 0.879
```

As it can be observed in the graphs in this table, the model run reach convergence in the range of 25-30 epochs.

The ROC-AUC or AUC for this model is independent of the threshold selected and has a value of 0.879.

This is not a perfect score. However, this implies a good classifier. (Close to excellent or AUC > 0.9)

Confusion matrix and classification report analysis:

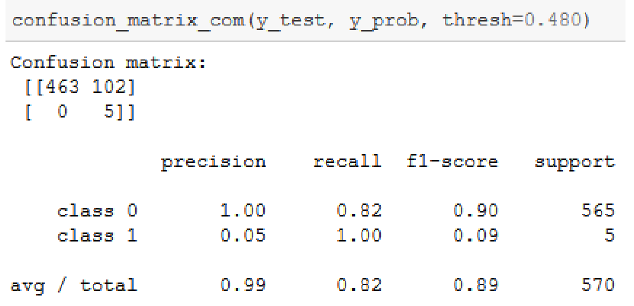


```
confusion_matrix_com(y_test, y_prob, thresh=0.5)

Confusion matrix:
 [[543  22]
 [  4   1]]

             precision    recall  f1-score   support

    class 0       0.99      0.96      0.98       565
    class 1       0.04      0.20      0.07         5

avg / total       0.98      0.95      0.97       570
```

Threshold = 0.5:

For this threshold, the model would classify correctly only on true exoplanet and it will miss 4 of the true exoplanets in the testing set. It would also produce 22 false positive classifications.

Total f1-score of 0.97.



```
confusion_matrix_com(y_test, y_prob, thresh=0.480)

Confusion matrix:
 [[463 102]
 [  0   5]]

             precision    recall  f1-score   support

    class 0       1.00      0.82      0.90       565
    class 1       0.05      1.00      0.09         5

avg / total       0.99      0.82      0.89       570
```

Threshold = 0.48:

For this threshold, the model would classify correctly the 5 true exoplanets. It would produce 102 false positives classifications.

Total f1-score of 0.89.

Figure 3. Confusion matrix and classification report analysis

Note: For the table above, class 0 means non-exoplanet and class 1 means exoplanet.

The Jupyter notebooks for the final model and second best are included in the folder included in the submission for this project:

1) exoplanet_Nov10_8cnn-ND_2dnn-ND_mse_Adam1e-5-betas-eps_relu_30epochs-v2.ipynb: This notebook includes the results from this section on the best model. (Second run) (AUC=0.879)
2) exoplanet_Nov10_8cnn-ND_2dnn-ND_mse_Adam1e-5-betas-eps_relu_30epochs.ipynb: First run of the best model. (AUC=0.882)
3) exoplanet_Nov09_8cnn_2dnn_mse_Adam1e-5_relu_30epochs.ipynb: Similar to best model except that it has dropout regularization. (AUC=0.77)

V. Conclusion

Conclusion: This is a good model for classification of exoplanets.

The final "best" model mentioned in the results sections have the ingredients for a good classifier already. It is important to mention that there is room for improvements.

The graphs from mode on MSE, loss and AUC for the best model and Confusion matrix and classification report analysis (Figure 3) from previous section support the claim of the conclusion above. Free-form visualization material was already presented in the previous section.

Reflection

The entire end-to-end problem solution included:

1) Understand the final objective of the problem.
2) Data exploration.
3) Identify algorithms and techniques to be used in the problem. This includes identifying metrics appropriate for the problem in question.
4) Preprocessing the data for model.
5) Identify parameters for the model. For deep learning, perform grid search of hyperparameters.
6) Execute runs and constantly evaluate performance of the model.
7) Return to step 4 if additional augmentation is required. If not go step 5.

Note: If we are increasing the size of the network or adding data augmentation, we night need to perform grid search again we might be dealing with a slightly different problem.

In retrospective:

1) Once that I started to follow a structure on the methodology for working with the problem, I was able to move and adjust faster with the project.
2) Once that I realized that I needed to perform a new grid search when I did a significant change to the network, I started moving faster and getting more stable models.

Improvement

Since there is a paper out there (reference 4) that points to a very successful implementation of their model a getting AUC over 0.97. It is clear that the model can have improvements. Their network was much bigger and they had availability to use the Google-Vizier that allowed them to test several thousand models. While I didn't have their computing resources, I was able to leverage some of their insights for this problem.

Some of the areas that perhaps could help on improving our model:

1) Using a much bigger network: This would take longer time and a few rounds of grid searches.
2) Some of the kernels found in Kaggle showed that some people use Fourier transform to work on frequency domain instead of time. Perhaps worth exploring for a future time.
3) Explore if filtering noise as a preprocessing step might bring some benefits. However, we have to investigate if removing noise would also remove information that the classifier would need. This could be investigated in a not so far future.

References

1) Wikipedia on exoplanets: https://en.wikipedia.org/wiki/Exoplanet
2) Exoplanet search dataset in kaggle: https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data/home
3) Github associated to (2): https://github.com/winterdelta/KeplerAI.
4) Paper on: IDENTIFYING EXOPLANETS WITH DEEP LEARNING: A FIVE PLANET RESONANT CHAIN AROUND KEPLER-80 AND AN EIGHTH PLANET AROUND KEPLER-90 (https://www.cfa.harvard.edu/~avanderb/kepler90i.pdf)
5) Metrics to Evaluate your Machine Learning Algorithm: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234
6) Metrics removed in Keras 2.0: f1, precision, and recall have been removed. Solution in stack overflow to use a custom metric function: https://stackoverflow.com/questions/43547402/how-to-calculate-f1-macro-in-keras
7) How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras, By Jason Brownlee on August 9, 2016 in Deep Learning: https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/
8) Gentle Introduction to the Adam Optimization Algorithm for Deep Learning, by Jason Browlee: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
9) The Area Under an ROC Curve: http://gim.unmc.edu/dxtests/roc3.htm