Machine Learning Engineer Nanodegree

Capstone Project Report

William Gilpin

Oct 2018

Udacity

One of the Great Questions of our time is that of the existence of non-terrestrial life. Of special relevance is the search for life from outside our solar system, as that would suggest a separate origin of that life, showing evolution has produced life at least twice, and therefore possibly many times. As part of this quest astronomers have been looking for planets with similar conditions to earth in terms of size, solidity and temperature range. Such planets will then become candidates for further investigation.

The Kepler mission was launched in 2009 in order to monitor 156,000 stars in the areas of Cygnus & Lyra with the intention of detecting earth-like planets in the stars' habitable zones (Borucki, 2010). The basic mechanism for detection is simple: monitor the light of a star over a period of time and look for periodic reduction in the brightness of the star for period in the range 2-16 hours (Johnson, 2015), this dimming being caused by the obscuration of a small fraction of the stars light during the planets transit between the observer and the star.

Problem Statement

The project is a two-class classification problem, to identify stars with planets given time-series data of stellar brightness from stars in the Kepler survey. The classification classes are "Planet Present" and "No Planet Present". The approach is to find periodic brightness variations in the range of 2-16 hours which would therefore suggest a transiting orbital body.

The input datasets being used are from NASA and were published for use in the Kaggle "Exoplanet Hunting" competition (Kaggle 2018). The output will be a single real number in the range [0.0 ... 1.0], representing the certainty of the presence of a planet.

Analysis

Basic Exploration

The dataset contains a small number of light curves from stars with a confirmed planet, and a large number of stars without. Observing the light curves there appears at first glance to be some periodicity to the positives, although there is also marked periodicity in some of the negative results too.

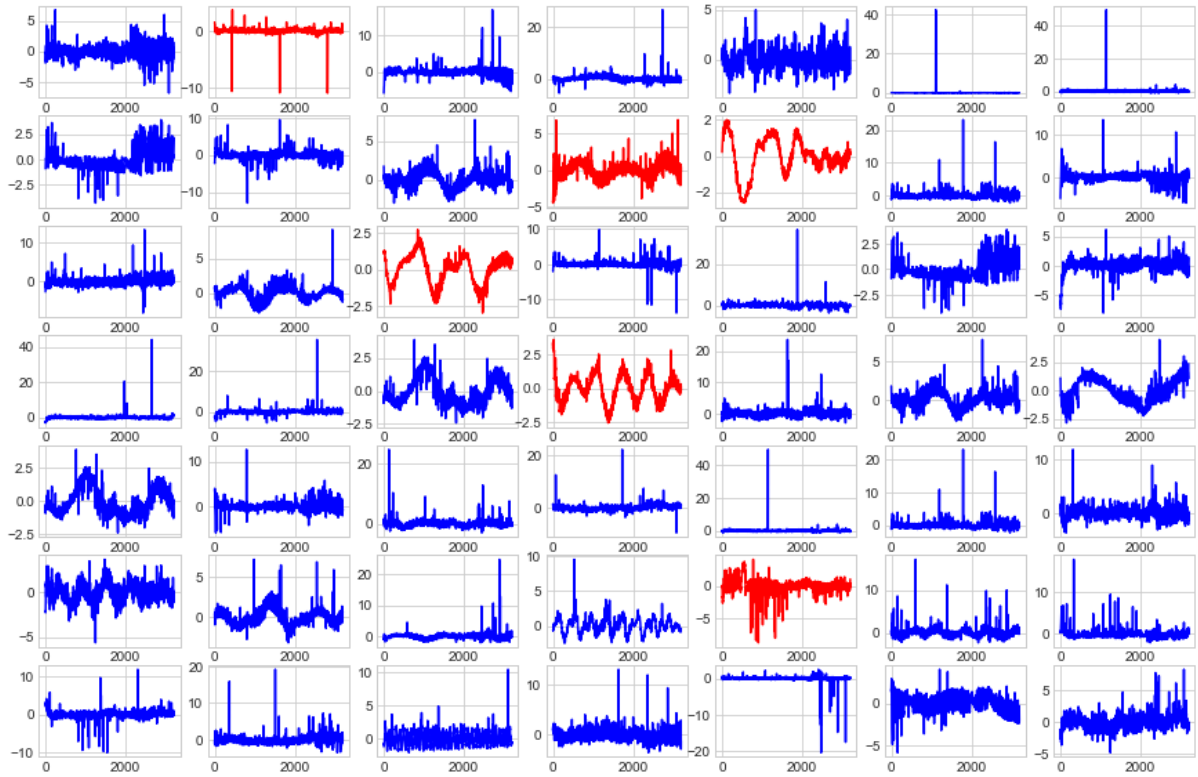


Fig. 1: Light curves of sample stars - red is a positive sample, blue is negative.

It can be seen there is a wide range in mean and variance of flux values between stars, so normalisation will be applied.

Algorithms & Techniques

A variety of approaches are used in this project, both in isolation and together. We use a convolutional neural network (CNN), a deep long-short-term memory (LSTM) network and Fast Fourier transforms (FFT). All three of these algorithms have a common goal of finding repeating patterns in time series data, which intuitively align with the project goal of identifying exoplanets by finding repeating patterns in the data.

Convolution Neural Networks

The primary algorithm used is a 1-dimensional convolutional neural network (CNN). In abstract this is a sliding window over the time-series data, trained by back propagation, to extract location invariant features. The most successful approach for the project was was found to be the use of a second CNN in a layer after the first one.

CNNs are inspired by neuroscientific models of the visual cortex (Hubel, 1962) describing the functional structure of the visual correct in terms of layers

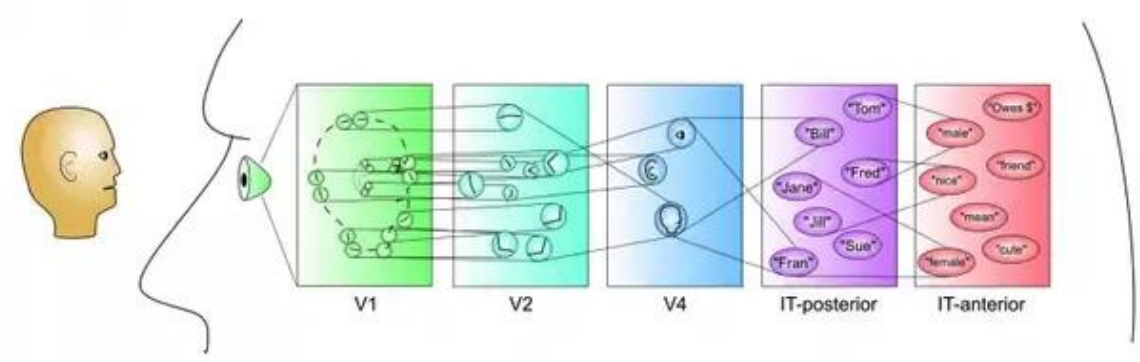


Fig. 2: Layers in the mammalian visual system. Image credit (Raval, 2018)

The combination of local connection, hierarchy and spatial invariance was developed by LeCun into the Convolutional Neural Network (LeCun, 1989). The operation of a CNN is to iteratively generate the dot product of a small weights matrix the sub-matrix described by a sliding window over one layer of the input matrix. This generates a value from each “convolution” which is the term for the sliding window, shown in green on the source matrix (Figure 2).

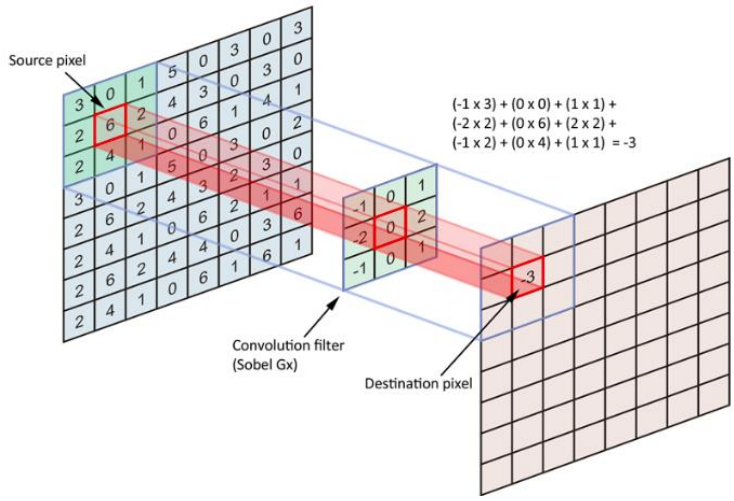


Fig. 3: Convolution as dot product

The dot product is therefore a value which indicates the degree to which the convolutional are of the source map matches the convolutional filter itself. If the filter represents a horizontal line leading to a 45 degree angle, then any regions of the source map containing a similar feature will lead to the output map having a high value for that region. This is expressed (Dettmers, 2015) as

$$feature\_map = input \otimes filter = \sum_{y=0}^{columns} \left(\sum_{x=0}^{rows} input(x-a, y-b)\, filter(x,y) \right)$$

The output matrix therefore smaller than the input matrix by a factor of $\frac{1}{size(filter)}$ in each dimension if the filter was moved by width(filter) each step, although this step value (**stride**) is a design parameter for the network. It is possible and desirable to reduce the size of the layer further by another filter called a **pooling** filter. A form of pooling called **max pooling** consists of identifying the max value in a given area of a matrix (eg a 2 x 2 sub matrix) and producing a new, smaller matrix where the max value represents the entire sub matrix. This final smaller matrix is of smaller spatial dimensions by $\frac{1}{size(pool)}$ in each dimension. Overall the convolution and pooling layers result in a feature map of substantially smaller dimensions than the source, and the feature map representing the extent to which the filter was correlated with each region on the source. The model will, however, require more than one filter, to enable detection of complex features, so although the original x- and y-dimensions of the input are reduced, each filter produces its own feature map, so the size in the z-dimension will increase accordingly.

The next step is regularisation (RELU) where each negative value in the feature map is set to 0, but positive values are unchanged: $\max(0,x)$ The final tool used in the convolutional part of the model is dropout where a specified percentage of the output values are randomly set to 0 (Hinton, 2012). The effect of dropout is to prevent overfitting - a particular worry given our very biased dataset.

The final network is then trained using gradient descent via back propagation, applying the partial derivative of the loss to each weight going backwards through the network.

Long Short Term Memory

The second algorithm used is a Long Short-Term Memory recurrent neural network.

A recurrent neural network (RNN) is a neural network where the output of one time-step is used as an input to the next time step. This allows RNNs to process sequential data in a way that takes account of the sequencing.

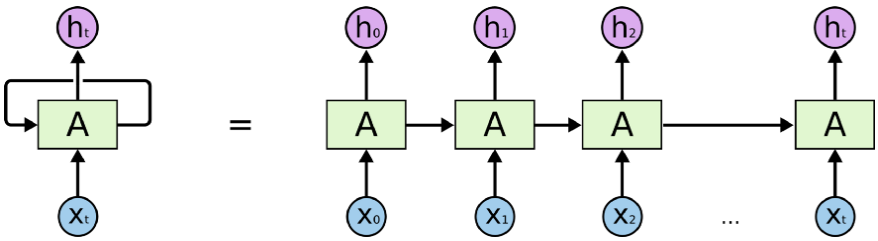


Fig. 4: Basic recurrent neural network (Image credit Ma, 2017)

Note the use of hidden state, h, to store the information to be passed to the next time-step. This is described formally as

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \theta)$$

That is, the hidden state **h** at time *t* is a function of

- the hidden state at time *t-1*, $\mathbf{h}^{(t-1)}$,
- the input **x** at time *t*, and
- the weights θ of the function.

RNNs of this form work well for short sequences (Bengio, 1994) but with long sequences a problem arises in gradient descent whereby for gradients less than 1 the repeated multiplication tends to zero, the **vanishing gradient problem**, and its corollary the **exploding gradient problem** where for gradients greater than 1, over a long dependent

sequence they tend to infinity.

Hochreiter & Schidhuber's (1997) solution is a network cell with a **forget gate**, controlling how much of previous state is allowed into the current cell. An LSTM cell has 3 separate networks/layers internally called gates, each with their own weights, and each differentiable and thus able to learn by back propagation:

- A forget gate, controlling how much of previous state is allowed into the cell.
- an input gate, controlling how much the current inputs affect the outgoing state from this cell.
- an output gate, producing a result incorporating state passed by the forget gate, and filtered by the input gate.

This LSTM architecture has been shown to perform well against longer sequences, but at a higher computational cost.

Fast Fourier Transforms

The final algorithm used is a Fast Fourier transform (FFT) (Dashnow, 2018). The FFT transforms a discrete array of time-domain waveform samples into an array of frequency-domain spectrum samples.

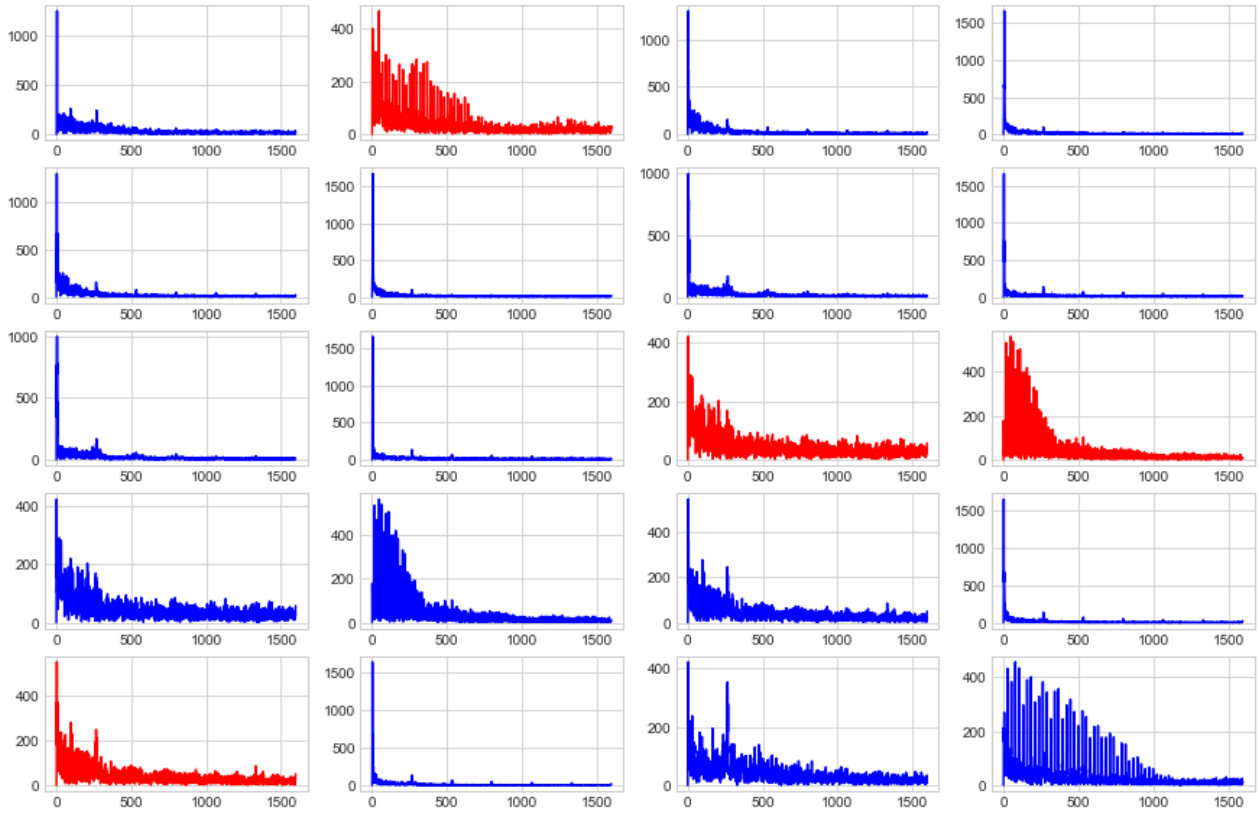
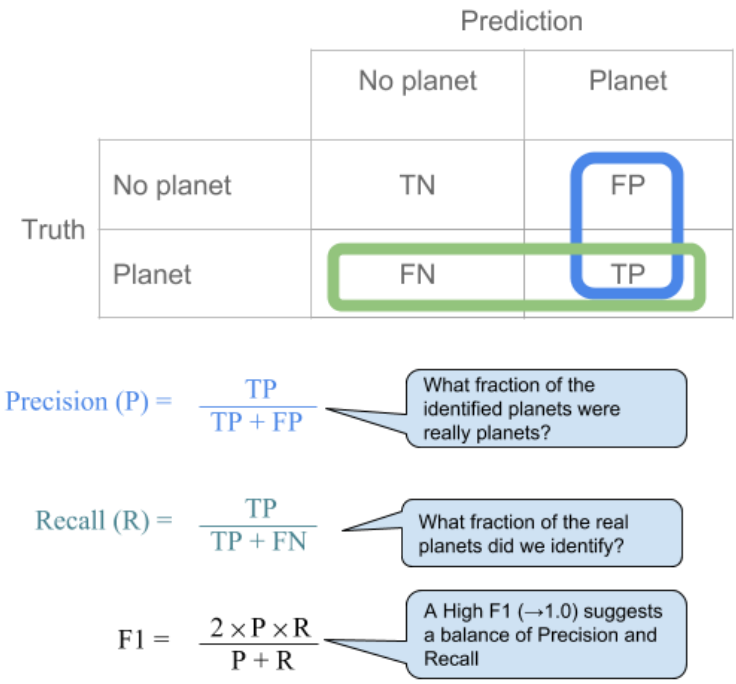


Fig. 5: Fourier analyses of sample of light curves. Red for +ve sample, blue -ve.

Simplistically the output is a histogram representing the strength of the frequency component of the input data for that frequency. For example the top left graph in figure 5 shows a strong low-frequency component (the spike at ≈ 0 Hz) and two further small spikes showing there is a small periodicity in the data at those frequencies.

F1 Score

The primary metric to be used is the F1 score:



Benchmark

For the benchmark we selected the Random Forest algorithm. Random forests are based on (Breiman, 2001). Random forest classifiers use the Gini impurity index, defined as

$$Gini(t) = \sum_{i=0}^{c-1} p_i(1-p_i)$$

For c classes, where p_i is the ratio of the class, i.e. the probability that a randomly chosen element would be incorrectly labelled if assigned a class based on the probability distribution of classes. This enables a recursive analysis of the data to generate an optimal decision tree, or in the case of a forest a collection of trees. For this project, computational cost limited the number of estimators to 100 - that is a forest of 100 trees, yielding the following results:

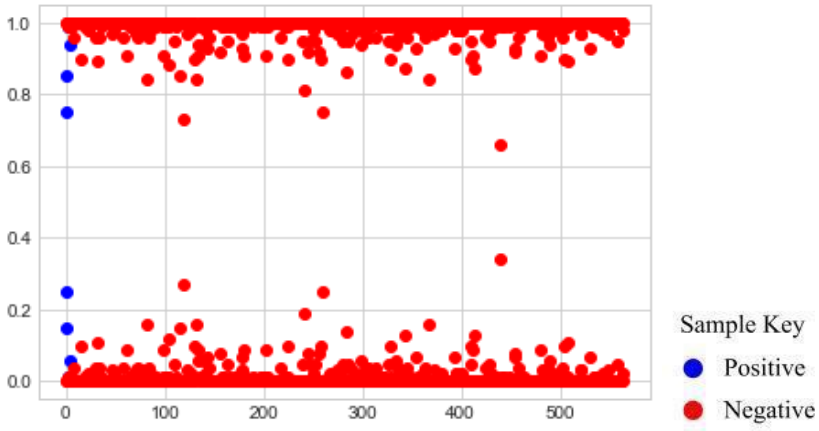


Fig. 6: Random forest classifier results.

The F1 analysis for the random classifier showed poor results, although fig. 6 suggests the positive samples were in fact weighted higher than the vast majority of negative samples, albeit not high enough for an accurate classification.

found:	0	
actual:	5	
missed:	5	
precision:	0.0	
recall:	0.0	
F1:	0.0	

Methodology

Data Pre-processing

The data has been cleaned by removing sensor artefacts, translating to a linear scale, removing likely cosmic ray artefacts, removing background flux and summing values over the photometric aperture of the target star image. They are then further pre-processed to allow for focusing and pointing artefacts, and for gaps in the time series (Jenkins 2018)

The files are presented as CSV files, one training set exoTrain and one test set exoTest. In the files there is 1 observation time-series per row. There are 5087 rows in the exoTrain file, and 570 rows in the exoTest file. Each row consists of a single label, LABEL, then 3197 flux points in columns FLUX.1 to FLUX.3197 representing flux values from t=1 to t=3197 for a single star

exoTest.csv (5.77 MB)

	# LABEL	# FLUX.1	# FLUX.2	# FLUX.3	# FLUX.4	# FLUX.5
1	2	119.88	100.21	86.46	48.68	
2	2	5736.59	5699.98	5717.16	5692.73	
3	2	844.48	817.49	770.07	675.01	
4	2	-826	-827.31	-846.12	-836.03	
5	2	-39.57	-15.88	-9.16	-6.37	
6	1	14.28	10.6299999999999	14.5599999999999	12.4199999999998	12.069999
7	1	-150.479999999996	-141.720000000001	-157.599999999999	-184.599999999999	-164.88999

For this project we generate training and validation datasets from the exoTrain dataset. Give the paucity of positive samples in the data, this will be manipulated to ensure a similar number of positives in each training set, by random sampling in the subsets of positive and negative samples. exoTest will be used as the test data.

The sampling interval for these long cadence files is a consistent 29.4 minutes or 56.7 mHz. The Nyquist frequency is 284 μHz [\[Murphy, 2018\]][MURPHY]. The variance of the flux measurements is wide and will need normalising.

Implementation

The models used in the projects are as follows.

FFT

Firstly, the raw light curves were subject to a Fourier Transform via the scipy fft module (scipy 2018). This produces a frequency domain distribution of the light curve,

symmetrical around the Nyquist frequency, so the FFT output is truncated in width at this point to discard the repeated (mirrored) data. Then the complex output is converted to magnitude, and this frequency data is passed through a deep LSTM network to identify cyclical features. This network is arranged thus:

```
Input Layer
LSTM(units=75)
LSTM(units=37)
Dense(units=25)
Dropout(0.5)
Dense(units=1)
```

The model has a total of 41,248 trainable parameters and was trained over 10 epochs due to high computational cost. The initial model had 1000 units per layer, but failed to train in 24 hours, so the number of LSTM hidden units was reduced until a reasonable training time was achieved (units=10) then gradually increased as shown. Another hyperparameter varied was the number of LSTM layers. Moving from 1 to 2 layers improved performance but further layers turned out to be very computationally expensive for little benefit. Once the model was move from a laptop to a Google Cloud GPU instance, we were able to add the dense layers shown and scale the number of LSTM hidden units.

CNN

The second architecture used was a deep 1-dimensional convolutional network consisting of 4 layers of convolutions and 2 final dense layers. The architecture is visualised as follows:

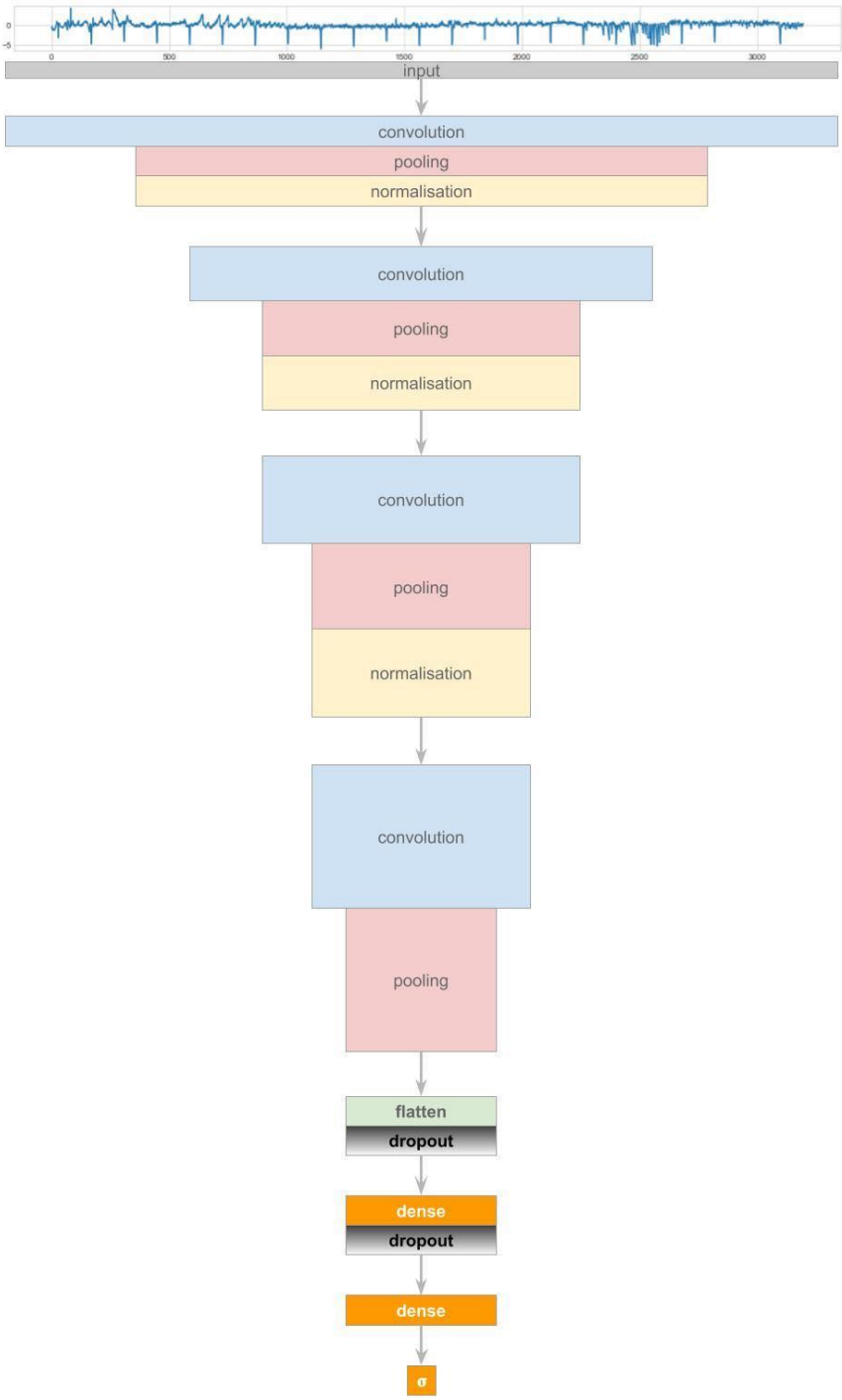


Fig. 7: Convolutional network architecture visualised

There are 4 instances of [convolution > pooling], and 2 dense layers. The model has 85,881 trainable parameters and is described formally in Annex B.

In order to train on a more balanced dataset a generator was used with 2 main aims: to ensure each training batch contains 50% positive samples, and to rotate those positive samples pseudo-randomly from batch to batch. The model is then trained for a few epochs at a high learning rate to encourage convergence, then the rate is reduced for the rest of the training run. To improve the accuracy, various hyperparameters and architectures were tried including:

- Increased number of CNN layers. We started with 2 and varied up to 5 which was too expensive. Accuracy improved up to 4 layers.
- Varying the number of filters. We started with 4 filters, doubled with each layer, and found accuracy improved slowly up to
- Increasing the kernel size - which yielded good results up to n=21
- Varying the dropout rates for the two Dense layers.

Composite Model

The final composite model is a 2-input model using the keras functional API:

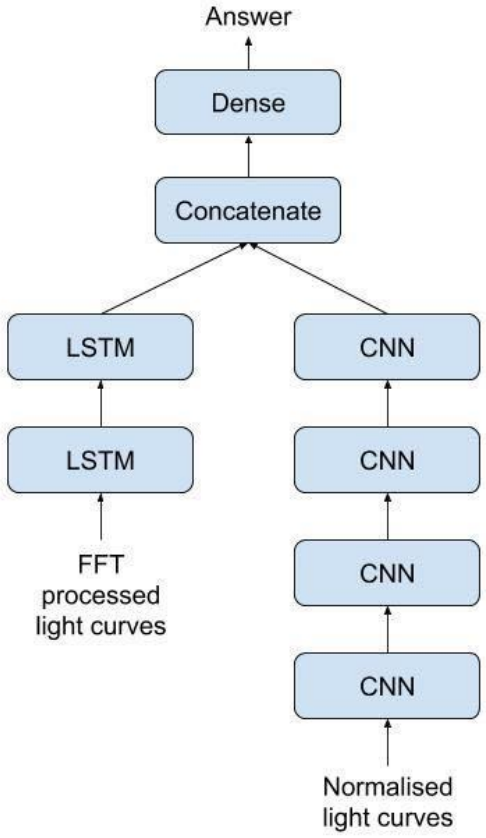


Fig. 8 Composite Network Overview

The model has 127, 240 trainable parameters and is formally described in Annex C. This model has two inputs, firstly the raw light curve data (normalised), and secondly the

output of the FFT processing on these curves. The model was not itself tuned, as the tuning was carried out on the two input branches.

Training Times

During training of the model, it was necessary to vary hyperparameters many times, and given that each epoch took 10+ minutes on a basic MacBook, the overall training times became unsustainable. As a result the decision was taken to move the training to a Google Cloud GPU backed instance, the "Deep Learning VM" specified as:

n1-highmem-2

2 x vCPU

13 GB RAM

2 x NVIDIA Tesla K80 GPU

100 GB persistent disk

In order to take advantage of the GPUs, the LSTM layer was replaced by CuDNNLSTM, which is the keras CUDA implementation of LSTM designed to run on a GPU. This improved training times for the LSTM & CNN layers by 10x. It was noted however that there was little or no training time improvement for the Random Forest classifier, as this module does not use the GPU.

Results

Model Evaluation and Validation

Evaluation: CNN model

The model trained well, with loss and accuracy both improving with number of training epochs. It is noticed however that cross-validation accuracy was slower to train

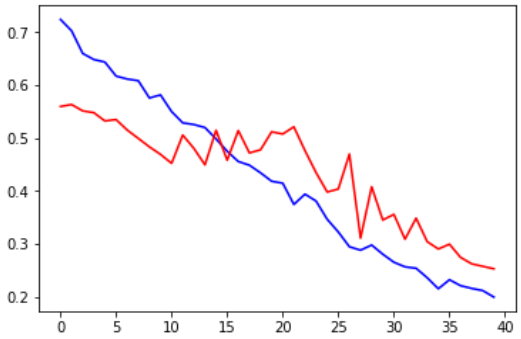


Fig. 9: Loss over training epochs. Blue for training data, red for cross-validation.

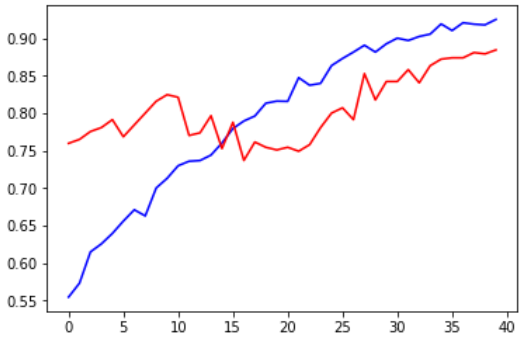


Fig. 10: Accuracy over training epochs. Blue for training data, red cross-validation.

Predictions were plotted from the raw data, as per 11. This demonstrates that positive samples generated high probability predictions.

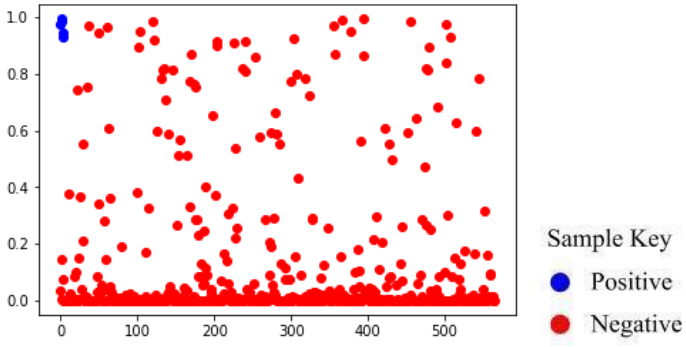


Fig. 11: Prediction scatter plot. The y-axis is predicted P(planet).

The crossover point was then calculated using scikit-learn's roc_curve metric module.

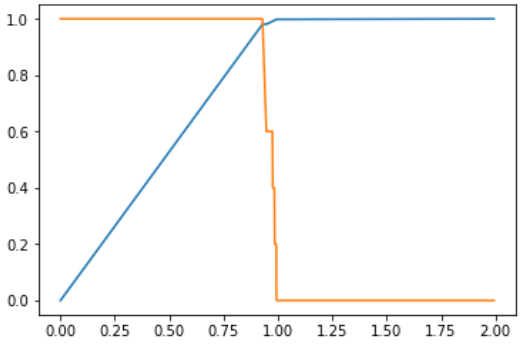


Fig. 12: Crossover point for prediction accuracy

Giving F1 score as follows:

found:	3
actual:	5
missed:	4
precision:	0.33
recall:	0.2
F1:	0.25

Evaluation: FFT model

The FFT model performed poorly in isolation. Training didn't converge, and the accuracy was low.

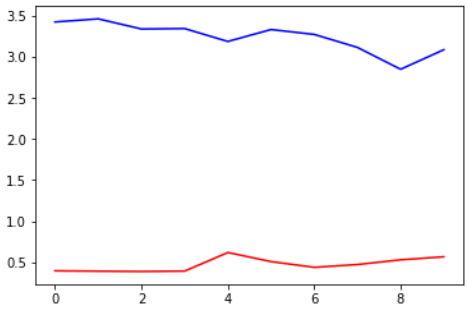


Fig. 13: Loss over training epochs. Blue for training data, red for cross-validation.

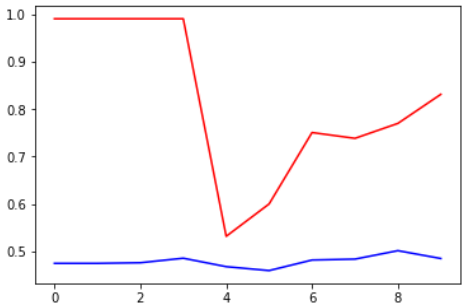


Fig. 14: Accuracy over training epochs. Blue for training data, red cross-validation.

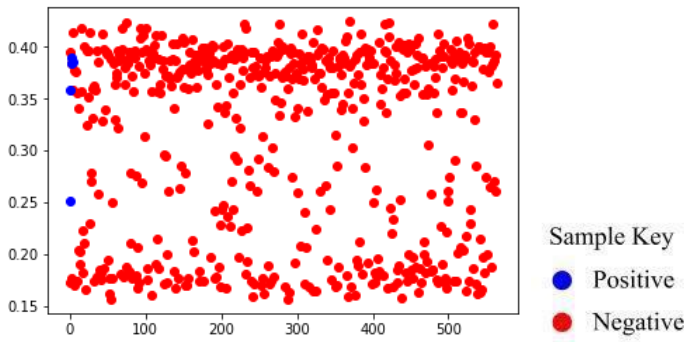


Fig. 15: Prediction scatter plot. The y-axis is predicted P(planet).

It can be seen that the performance of the cross validation data never converged, leading to a poor F1 score:

found:	570
actual:	0
missed:	0
precision:	0.0
recall:	0.0
F1:	0.0

Evaluation: composite model

The composite model performed much better than either of the individual models, with an F1 score of 0.53:

found:	10
actual:	5
missed:	1
precision:	0.4
recall:	0.8
F1:	0.53

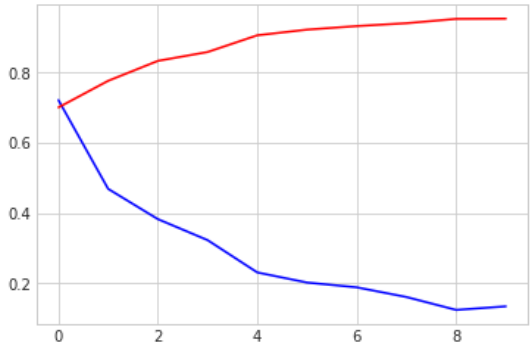


Fig. 16 Loss(blue) and accuracy (red) per epoch.

Despite the apparently poor results of the FFT model, which therefore might be expected to have contributed little to the final result, the results are better. From the scatter plot (Figure 16) it is clear that there is good differentiation between positive and negative samples is happening as this is a very imbalanced dataset yet as many of the positive samples as negative ones have P(planet) > 0. There is no clear separability, but the model found 2 false positives for every 1 true positive which is a big improvement.

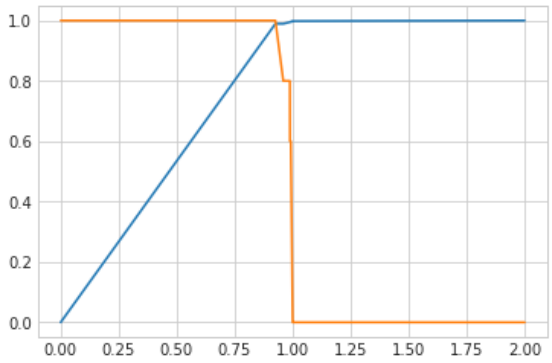


Fig. 17: Crossover analysis gives a crossover at 0.93

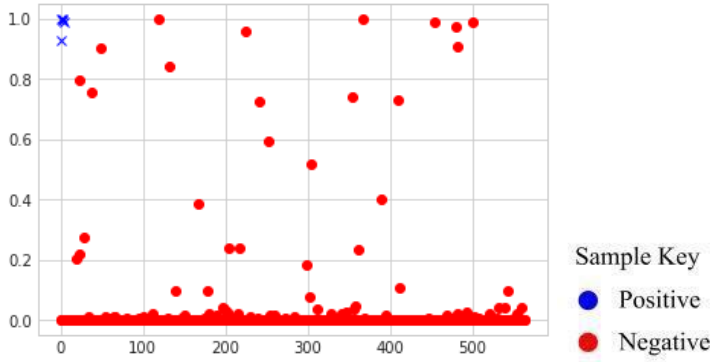


Fig. 18: Prediction scatter plot. The y-axis is predicted P(planet).

Justification

The overall performance of the model is not perfect, but much better than the benchmark or any of the sub-models, with F1 scores as follows:

- FFT model alone: F1 score 0.00
- CNN model alone: F1 score 0.25
- Composite: F1 score 0.53
- Benchmark: F1 score 0.00

The 2:1 ratio of false positives to true positives would certainly narrow down the task

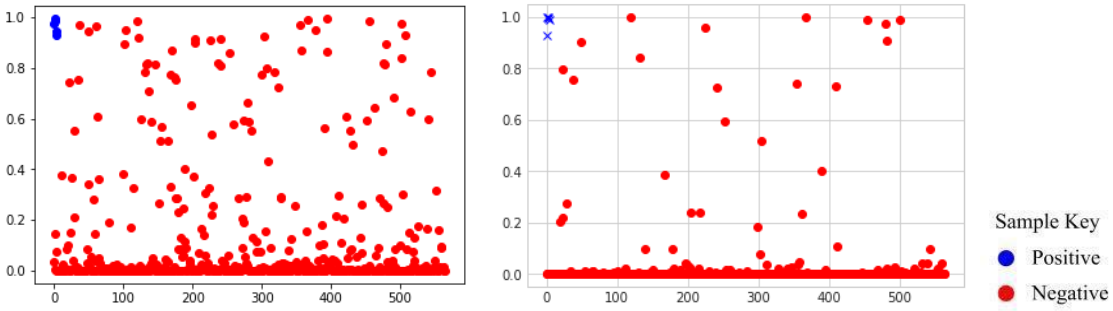
of a researcher seeking planets, compared to 500:1 in the test data.

Although it would not be accurate to say this model has completely solved the problem, it clearly offers value and it seems likely further improvements could be made.

Conclusion

Reflections

The goal was to analyse the Kepler satellite light curves for 5,086 stars in the Kepler Survey, in order to identify likely exoplanet candidates. The implemented solution was a 2-input model built with the keras functional API. The inputs were firstly 2 layers of LSTM processing FFT frequency-domain spectrum analysis of the sample data, and secondly four layers of 1-dimensional CNNs processing the raw light curves. These two networks were combined in a final fully-connected layer to produce a prediction. Of interest was controlling the complexity of the model to avoid excessive computational load, achieved by using the 2-input model. The single most striking aspect of the project though was that the model is clearly differentiating to some extent, as emphasised by both figures 11 and 18 which show positive samples in the top of the prediction range. This suggests that with further model tuning and refinement a better result could be obtained.



Figures 11 and 18 repeated for convenience

Note that in both instances the positive samples achieve a much higher average P(planet) than the negative ones. Although there is no separation between the two populations, it is clear that some progress towards accurate classification is being made.

Possible Improvements

Specific improvements should be focused on the FFT/LSTM model. Modifications to the model produced large end-to-end improvements in the final model. Although it produced

poor results itself, it can be seen by comparing Figure 11 and Figure 16 that the FFT branch in the composite model did in fact reduce false positives. Areas to focus on would include:

- processing the complex number outputs of the FFT - we currently use the magnitude, but could use both real and imaginary parts,
- the number of hidden units in the LSTM layers,
- The number of LSTM layers.

Another area for consideration would then be the fully-connected layers in the composite model. Currently there is only a single layer, but these could be expand to assist in feature integration form the two branches.

References

Dashnow, H., van der Walt, S., & Nunez-Iglesias, J. (n.d.). 4. Frequency and the Fast Fourier Transform - Elegant SciPy [Book]. Retrieved 13 October 2018, from https://www.oreilly.com/library/view/elegant-scipy/9781491922927/ch04.html

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166. https://doi.org/10.1109/72.279181

Borucki, W. J., Koch, D. G., Basri, G., Batalha, N., Brown, T. M., Bryson, S. T., … Still, M. (2011). Characteristics of Planetary Candidates Observed by Kepler. II. Analysis of the First Four Months of Data. *The Astrophysical Journal*, *736*(1), 19. https://doi.org/10.1088/0004-637X/736/1/19

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Dettmers, T. (2015, March 26). Understanding Convolution in Deep Learning. Retrieved 11 October 2018, from http://timdettmers.com/2015/03/26/convolution-deep-learning/

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv:1207.0580 [Cs]*. Retrieved from http://arxiv.org/abs/1207.0580

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, *160*(1), 106–154. https://doi.org/10.1113/jphysiol.1962.sp006837

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Jenkins, J. M., Caldwell, D. A., Chandrasekaran, H., Twicken, J. D., Bryson, S. T., Quintana, E. V., Borucki, W. J. (2010). OVERVIEW OF THE KEPLER SCIENCE PROCESSING PIPELINE. *The Astrophysical Journal*, *713*(2), L87–L91. https://doi.org/10.1088/2041-8205/713/2/L87

Jenkins, J. M. (2017). *KEPLER DATA PROCESSING HANDBOOK*. NASA Ames Research Center. Retrieved from https://archive.stsci.edu/kepler/manuals/KSCI-19081-002-KDPH.pdf

Johnson, M. (2015, April 13). Mission overview [Text]. Retrieved 7 October 2018, from http://www.nasa.gov/mission_pages/kepler/overview/index.html

Kaggle, 2018. Exoplanet Hunting in Deep Space. (n.d.). Retrieved 7 October 2018, from

https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541–551.

Lemaitre, G., Nogueira, F., Oliveira, D. and Aridas, C. (2016). Over-sampling — imbalanced-learn 0.3.0 documentation. Retrieved 28 July 2018 from https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html.

Ma, J. (2016, April 2). All of Recurrent Neural Networks. Retrieved 11 October 2018, from https://medium.com/@jianqiangma/all-about-recurrent-neural-networks-9e5ae2936f6e

Murphy, S. J. (2012). An examination of some characteristics of Kepler Short and Long Cadence Data. *Monthly Notices of the Royal Astronomical Society*, *422*(1), 665–671. https://doi.org/10.1111/j.1365-2966.2012.20644.x

Raval, S. (2018). *Code for 'Convolutional Neural Networks: The Math of Intelligence (Week 4)' By Siraj Raval on Youtube - llSourcell/Convolutional_neural_network*. Jupyter Notebook. Retrieved from https://github.com/llSourcell/Convolutional_neural_network (Original work published 2017)

Russell, S. J., Norvig, P., & Davis, E. (2010). *Artificial intelligence: a modern approach* (3rd ed). Upper Saddle River: Prentice Hall.

scipy.fftpack.fft — SciPy v1.1.0 Reference Guide. (n.d.). Retrieved 13 October 2018, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.fftpack.fft.html#scipy.fftpack.fft

Appendix A

LSTM model structure

```
fft_inputs = Input(shape=fft_input_shape, name='main_input')
x = CuDNNLSTM(75, return_sequences=True, name='L1')(fft_inputs)
x = CuDNNLSTM(37, name='L2')(x)
x = Dense(25, name='M1')(x)
fft_final = Dropout(0.5)(x)
output = Dense(1, name='D1')(fft_final)
fft_model = Model(fft_inputs, output)
fft_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
main_input (InputLayer)      (None, 1599, 1)           0
_________________________________________________________________
L1 (CuDNNLSTM)               (None, 1599, 75)          23400
_________________________________________________________________
L2 (CuDNNLSTM)               (None, 37)                16872
_________________________________________________________________
M1 (Dense)                   (None, 25)                950
_________________________________________________________________
dropout_5 (Dropout)          (None, 25)                0
_________________________________________________________________
D1 (Dense)                   (None, 1)                 26
=================================================================
Total params: 41,248
Trainable params: 41,248
Non-trainable params: 0
_________________________________________________________________
```

Appendix B

CNN model structure

```
# input layer
conv_inputs = Input(shape=x_train.shape[1:])

# 1st feature extractor
conv1 = Conv1D(filters=8, kernel_size=21, activation='relu')(conv_inputs)
pool1 = MaxPool1D(strides=4)(conv1)
batch1 = BatchNormalization()(pool1)

# 2nd feature extractor
conv2 = Conv1D(filters=16, kernel_size=21, activation='relu')(batch1)
pool2 = MaxPool1D(strides=4)(conv2)
batch2 = BatchNormalization()(pool2)

# 3rd feature extractor
conv3 = Conv1D(filters=32, kernel_size=21, activation='relu')(batch2)
pool3 = MaxPool1D(strides=4)(conv3)
batch3 = BatchNormalization()(pool3)

# 4th feature extractor
conv4 = Conv1D(filters=64, kernel_size=21, activation='relu')(batch3)
pool4 = MaxPool1D(strides=4)(conv4)

# flatten feature extractors
flat1 = Flatten()(pool4)
drop1 = Dropout(0.5)(flat1)

# interpretation layers
dense1 = Dense(64, activation='relu')(drop1)
drop2 = Dropout(0.25)(dense1)
dense2 = Dense(64, activation='relu')(drop2)

# prediction
output_layer = Dense(1, activation='sigmoid')(dense2)

#model
f_model = Model(inputs=conv_inputs, outputs=output_layer)
```

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 3197, 2)	0
conv1d_13 (Conv1D)	(None, 3177, 8)	344
max_pooling1d_13 (MaxPooling	(None, 794, 8)	0
batch_normalization_10 (Batc	(None, 794, 8)	32
conv1d_14 (Conv1D)	(None, 774, 16)	2704
max_pooling1d_14 (MaxPooling	(None, 194, 16)	0
batch_normalization_11 (Batc	(None, 194, 16)	64
conv1d_15 (Conv1D)	(None, 174, 32)	10784
max_pooling1d_15 (MaxPooling	(None, 44, 32)	0
batch_normalization_12 (Batc	(None, 44, 32)	128
conv1d_16 (Conv1D)	(None, 24, 64)	43072
max_pooling1d_16 (MaxPooling	(None, 6, 64)	0
flatten_4 (Flatten)	(None, 384)	0
dropout_7 (Dropout)	(None, 384)	0
dense_17 (Dense)	(None, 64)	24640
dropout_8 (Dropout)	(None, 64)	0
dense_18 (Dense)	(None, 64)	4160
dense_19 (Dense)	(None, 1)	65

Total params: 85,993
Trainable params: 85,881
Non-trainable params: 112

Appendix C

Composite model structure

```
concatenated = concatenate([fft_final, dense2])
answer = Dense(1, activation='sigmoid')(concatenated)
composite_model = Model([conv_inputs, fft_inputs], answer)
composite_model.summary()
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 3197, 2)	0	
conv1d_5 (Conv1D)	(None, 3177, 8)	344	input_2[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 794, 8)	0	conv1d_5[0][0]
batch_normalization_4 (BatchNor	(None, 794, 8)	32	max_pooling1d_4[0][0]
conv1d_6 (Conv1D)	(None, 774, 16)	2704	batch_normalization_4[0][0]
max_pooling1d_5 (MaxPooling1D)	(None, 194, 16)	0	conv1d_6[0][0]
batch_normalization_5 (BatchNor	(None, 194, 16)	64	max_pooling1d_5[0][0]
conv1d_7 (Conv1D)	(None, 174, 32)	10784	batch_normalization_5[0][0]
max_pooling1d_6 (MaxPooling1D)	(None, 44, 32)	0	conv1d_7[0][0]
batch_normalization_6 (BatchNor	(None, 44, 32)	128	max_pooling1d_6[0][0]
conv1d_8 (Conv1D)	(None, 24, 64)	43072	batch_normalization_6[0][0]
max_pooling1d_7 (MaxPooling1D)	(None, 6, 64)	0	conv1d_8[0][0]
main_input (InputLayer)	(None, 1599, 1)	0	
flatten_1 (Flatten)	(None, 384)	0	max_pooling1d_7[0][0]
L1 (CuDNNLSTM)	(None, 1599, 75)	23400	main_input[0][0]
dropout_3 (Dropout)	(None, 384)	0	flatten_1[0][0]
L2 (CuDNNLSTM)	(None, 37)	16872	L1[0][0]
dense_1 (Dense)	(None, 64)	24640	dropout_3[0][0]
M1 (Dense)	(None, 25)	950	L2[0][0]
dropout_4 (Dropout)	(None, 64)	0	dense_1[0][0]
dropout_5 (Dropout)	(None, 25)	0	M1[0][0]
dense_2 (Dense)	(None, 64)	4160	dropout_4[0][0]
concatenate_2 (Concatenate)	(None, 89)	0	dropout_5[0][0] dense_2[0][0]
dense_5 (Dense)	(None, 1)	90	concatenate_2[0][0]

```
Total params: 127,240
Trainable params: 127,128
Non-trainable params: 112
```