

# Finding flares in Kepler and TESS data with recurrent deep neural networks

Krisztián Vida<sup>1</sup>, Attila Bódi<sup>1,3</sup>, Tamás Szklenár<sup>1</sup>, and Bálint Seli<sup>1,2</sup>

<sup>1</sup> Konkoly Observatory, Research Centre for Astronomy and Earth Sciences, Eötvös Loránd Research Network (ELKH), Konkoly Thege Miklós út 15-17, H-1121 Budapest, Hungary  
e-mail: [vidakris@konkoly.hu](mailto:vidakris@konkoly.hu)

<sup>2</sup> Eötvös Loránd University, Pázmány Péter sétány 1/A, Budapest, Hungary

<sup>3</sup> MTA CSFK Lendület Near-Field Cosmology Research Group

Received September 15, 1996; accepted March 16, 1997

## ABSTRACT

Stellar flares are an important aspect of magnetic activity – both for stellar evolution and circumstellar habitability viewpoints – but automatically and accurately finding them is still a challenge to researchers in the Big Data era of astronomy. We present an experiment to detect flares in space-borne photometric data using deep neural networks. Using a set of artificial data and real photometric data we trained a set of neural networks, and found that the best performing architectures were the recurrent neural networks (RNNs) using Long Short-Term Memory (LSTM) layers. The best trained network detected flares over  $5\sigma$  with  $\geq 80\%$  recall and precision and was also capable to distinguish typical false signals (e.g. maxima of RR Lyr stars) from real flares. Testing the network trained on Kepler data on TESS light curves showed that the neural net is able to generalize and find flares – with similar effectiveness – in completely new data having previously unseen sampling and characteristics.

**Key words.** Methods: data analysis– Stars: activity– Stars: flare– Stars: late-type

## 1. Introduction

Stellar flares are the result of magnetic field line reconnection – they appear as sudden brightening of the stellar atmosphere as a portion of magnetic energy is released. Flares are most common in late-type M-dwarfs (Walkowicz et al. 2011), but they appear in a wide range of main-sequence stars, and even in some evolved giant stars (Oláh et al. 2021). These eruptions are interesting not only from a stellar astronomy viewpoint: they can also influence their surroundings, and have serious effects on the atmosphere evolution of their orbiting planets (Khodachenko et al. 2007; Yelle et al. 2008; Vida et al. 2017).

Recent and upcoming photometric space missions, like Kepler (Borucki et al. 2010), TESS (Ricker et al. 2015) and PLATO (Rauer et al. 2014) provide an unprecedented opportunity for astronomers to study these events. But this opportunity comes with a new challenge – while earlier observations of a single object could be handled relatively easily by manual analysis, investigating photometric data of hundreds, or even thousands of targets manually is not feasible.

There are several existing approaches to detect these transient events in space automatically. These are often based on smoothing the light curve and detecting the outliers by some criteria (Davenport 2016; Stelzer et al. 2016), or other way of outlier detection (e.g. the RANSAC method, see Vida & Roetenbacher 2018). A known issue of these methods is that they are prone to misidentify other astrophysical phenomena as flares (e.g., maxima of KIC 1572802, an RR Lyræ star), and they also need user attention.

These issues can be possibly remedied by deep learning methods – a neural network, once trained by appropriate training data, does not need any user input or parametrization, and

can operate autonomously on new data, and hopefully, the "astrophysical noise" (like RR Lyr peaks marked as flares) can be also filtered out. While the training of such networks is time consuming – a single iteration can have a runtime of several hours even on recent GPUs – analysis of new data is relatively fast: for example, face recognition can be done on the fly in video streams on mobile phones.

Using simple dense neural networks would be impractical for this case, since that approach would yield too many variables, making the algorithm too slow. For time series data Convolutional Neural Networks (CNNs, LeCun et al. 1999) are a much better choice, as these are much faster or better adapted for such data. Convolutional networks take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Recently, Feinstein et al. (2020) presented a method to find flares using CNNs.

Another approach is using Recurrent Neural Networks (RNNs): here, connections between nodes form a directed graph along a temporal sequence. This allows them to "remember" their previous state, making them suitable for a wide range of problems from improved machine translation (Sutskever et al. 2014) to anomaly detection in ECG time signals (Oh et al. 2018). This can give them the power not just simply detect outlying points, but also compare the analyzed data to previously seen light curve parts, just as a human would do, making it a powerful tool for scientists.

In this paper we present our first results of a flare detection method based on recurrent neural networks. In Sect. 2. we show the data preparation steps, in Sect. 3. the design and in Sect. 4. the evaluation of tested networks. In Sect. 5-6. we present the validation steps, while in Sec. 7. we discuss the performance of

our network. We dedicate Sect. 8 to present out dead ends and finally, we summarize our results in Sect. 10.

## 2. Input data

Our training data consisted of two sets: artificial data and real Kepler observations. For the artificial data we generated light curves of spotted stars using PyMacula (Kipping 2012)<sup>1</sup>. The artificial light curves were generated with different rotation periods. The periods of the light curves were chosen from a lognormal distribution with a mean of log 5 days, and the lowest period was set to 0.4 days. This distribution was selected to represent the most active, fast-rotating stars that are more likely to show flares. To these light curves a randomly selected number of spots (between 2–12) were added with spot evolution enabled. Then, we added noise with normal distribution with a noise level of  $8 \times 10^{-3} - 4 \times 10^{-5}$  (in normalized flux units), that correspond to typical short-cadence noise levels for 7–16 magnitude stars<sup>2</sup>. These light curves were injected with flares using the analytical flare model of Davenport (2016), using FWHM values between 0.5–1 hour with uniform distribution and amplitudes with lognormal distribution with a mean of  $10^{-5}$  and a standard deviation of 1. The number of injected flare per star varied between 5 and 200. The code `ocelot-gym` used to create spotted light curves injected with flares for the training is available on GitHub.<sup>3</sup>

Real Kepler observations with flares were selected from the targets of Roettenbacher & Vida (2018) with additional targets as ‘astrophysical noise’ (RR Lyr, Cepheids, flaring and non-flaring eclipsing binaries, systems with exoplanets, irregular variable stars). The selected short cadence light curves were first flagged with the FLATW’RM code (Vida & Roettenbacher 2018), and these flagged data were checked twice manually and re-flagged if needed. This set consisted of 214 light curves, that were separated to ‘active’ (107) and ‘quiet’ (207) sets for the first steps of the model selection. For the training we used only short-cadence data, since interpolating the long-cadence light curves would add too much information.

We found that using both time and intensity data as input the candidate neural nets did not converge, therefore we interpolated our data to 1-min cadence, and used only the flux values as a one-dimensional input vector. This, on one hand, helps some issues with the time axis (e.g. normalization), and also makes the analysis faster. On the other hand, this limits the usage of the trained network: while a data with 5-min cadence might be still handled fine, interpolating a long-cadence observation with 30-minute sampling adds too much information to the data, and also renders the analysis unnecessarily slow, therefore, for such data a different training will be needed. Gaps in the data were interpolated and filled with random noise with normal distribution – with characteristics of the light curve – to avoid jumps in the light curve that can mimic flares.

Flux data were also standardized, after dropping NaN non-numeric values, the observed flux was first divided by its truncated mean, then centered around zero by subtracting 1, finally we divided by the truncated peak-to-peak variation. The truncated mean and peak-to-peak variation were calculated after dropping flux values above the 99.9th percentile of the data (leaving out large flares). Experience shows that machine learn-

**Table 1.** Parameters of the tested network architectures.

Parameter	Tested values	Chosen value
Kernel	[GRU, LSTM]	LSTM
Kernel units	[128, 256]	128
Number of recurrent layers	[2, 3]	3
Dense layers	[0, 1]	0
Dropout rate	[0.2, 0.5]	0.2
Optional dense layers (units, activation)	512, sigmoid 64, sigmoid	
Output dense layer (units, activation)	1, sigmoid	
Loss function	Binary crossentropy	
Optimizer	NAdam (learning rate=0.001)	
Metrics	Accuracy, Recall	

ing algorithms perform best with standardized data having a variation more or less 1.

The prepared light curves were concatenated and used as a single one-dimensional input vector for the training. Training can be made faster with training multiple batches simultaneously (the batch number is mainly limited by GPU memory). However, creating batches with time series is not obvious, as the consecutive data parts are not independent from each other. Therefore – as opposed to the default time series generator in Keras – we created the batches by separating the data into  $b$  consecutive data segments, that feeds these batches after each other into the network for training. Using too large batch number might hinder the convergence of the network (especially if the length of the resulting batches are in the magnitude of the events we are looking for), but we found a batch number of  $b = 2048$  a good compromise. With this setup, the runtime of a single epoch was 12–17 minutes on an NVidia GeForce RTX 2080 Ti graphics card.

For the analysis each light curve point was associated with a flag to mark if it is part of a flare or not. This flag was determined based on 64-point windows starting from the given data point: the labels were calculated as the average of the input flags rounded to 0 or 1. In the case of the final data points, where the number of the remaining points is less than 64, the original data points were mirrored in order to avoid losing any data. Our experiments with different window sizes and different ways of calculating the labels, e.g. including exceptions for short events, or aligning the windows differently, all yielded poorer results (see also Sect. 8).

After selecting the most successful network candidate we returned to the input: we used its output to re-examine the training data. We checked the output for possible missed events that was not marked as flares in the original set, or light curve regions we marked incorrectly as flares. The network was then re-trained with this refined training set. These corrections improved the performance (recall/accuracy) up to 10% in the case of smaller events.

## 3. Network architecture

The neural network was built using Keras<sup>4</sup>, a deep learning API built on Tensorflow<sup>5</sup>. We experimented with a large number of network architectures (see Sect. 8), but our final network candidates consisted of sequential models with two or three GRU/LSTM layers with 128/256 units (Gated Recurrent Unit,

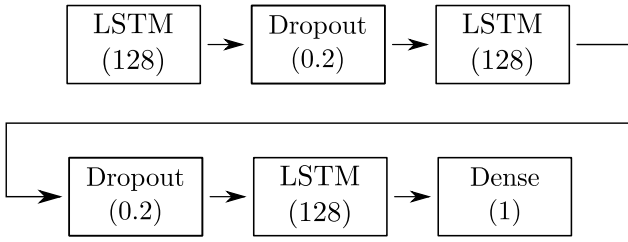
<sup>1</sup> <https://pypi.org/project/pymacula/>

<sup>2</sup> <https://nexsci.caltech.edu/workshop/2012/keplergo/CalibrationSN.shtml>

<sup>3</sup> <https://github.com/vidakris/flatwrm2>

<sup>4</sup> [keras.io](https://keras.io)

<sup>5</sup> <https://www.tensorflow.org>



**Fig. 1.** The final neural network architecture consisted of three LSTM layers with 128 units each, and a dropout layer between them with dropout rate of 0.2. The output was generated by a one-unit dense layer with sigmoid activation function.

Cho et al. 2014; Long Short-Term Memory layer, Hochreiter & Schmidhuber 1997). Between the recurrent layers a dropout layer (Srivastava et al. 2014) was added with a dropout rate of 0.2 – utilizing such layers in a networks serves as a kind of regularization, by temporarily removing a fraction of the neurons randomly from the network. This way our model can achieve lower generalization error, thus prevent overfitting, at a cost of somewhat increased computation time. After the recurrent layers a single one-unit dense layer was added with sigmoid activation to generate the final output (Fig. 1). The networks were optimized using a binary crossentropy loss function and an NAdam optimizer (Kingma & Ba 2014; Dozat 2015). As the ratio of flaring and non-flaring data points is far from equal, weighting is important during the fit, so the model pays more attention to samples from the under-represented class – we weighted the flaring points with their ratio to the total number of points.

The first stage of the model selection was performed using the HParams tool of Keras, which allowed us to compare a variety of model architectures. The options tried in this step of the hyperparameter tuning are summarized in Table 1 (see also Appendix A for more details). In this stage we trained the networks on our artificial data and the flaring Kepler data set. From the best performing models the first five best-performing networks were selected during hyperparameter tuning. Then, we added the non-flaring ‘astrophysical noise’ observations (see Fig. 2) to the training sample to find a network that not only can detect flares, but also neglect irrelevant signals. While with the flaring data the GRU networks performed well, adding the non-flaring data produced a large number of false positive detections.

## 4. Evaluation

After training the candidate networks, we ran a prediction for the validation data set, and compared them based on their accuracy (the rate of correctly classified points), classification error rate (rate of incorrectly classified points), recall (ratio of recovered flare points), precision (ratio of correct detection in the selected points), false positive rate, and  $F_\beta$  values:

$$F_\beta = \frac{(1 + \beta^2) \times \text{TP}}{(1 + \beta^2) \times \text{TP} + \beta \times \text{FN} + \text{FP}}.$$

Here TP, FN, and FP stand for true positives, false negatives and false positives, respectively, and  $\beta = 1$ .  $F_\beta$  score gives the weighted harmonic mean of precision and recall. We found, that the best-performing networks were the two-layered LSTM(256) with or without additional CNN layers, and the three-layered LSTM(128) networks. We selected the latter candidate as the final choice, as its runtime was  $\approx 20 - 40\%$  lower (see Appendix A).

Evaluating the performance of recurrent neural networks (RNN) is not as straightforward as in case of e.g. binary classifiers, where usually we have a set of train and test data with an input label for each item, which can be easily compared to the output label with the highest probability (for the complexity of the task see e.g. Arras et al. 2019). Particularly, in case of the flare detection using temporal information the problem comes from the lack of solid ground truth. First, we have to decide which kind of phenomenon is considered to be a flare and have to decide which points are part of that flare like event and which are not. As a possible solution we split the light curves and fit each input and predicted segment with a model individually and use the fits as “labels”. The fitting procedure is described in Sect. 6. In the following, we bin the flares by their signal-to-noise ratio and relative amplitude and calculate the aforementioned metrics for each bin, and compare the resulted curves instead of single numbers. As it is nearly impossible to force the RNN to select the same points for a given flare that we flagged as input, the fitting may result in different number of flares within the same event if its structure is complex. Therefore we consider input-output flares the same if they lie within 0.02 days.

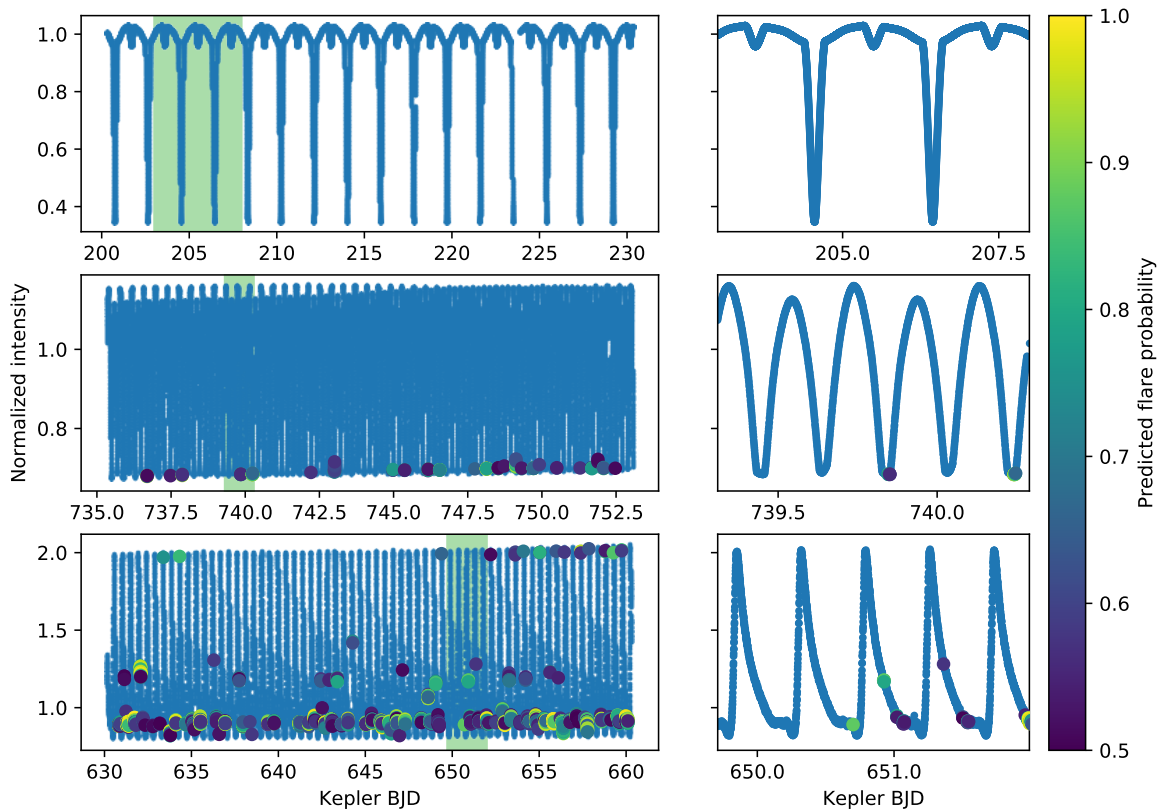
## 5. K-Fold cross-validation

To check the robustness of the neural network, we performed K-Fold cross-validation on the training data. This method is commonly used in applied machine learning to compare models and to see how dependent the trained network is on the particular set of training data. The K-Fold cross-validation method is based on separating a set of test data (we used 10% of the total data), and shuffling the remaining data into non-overlapping training and validation sets multiple times (we used 5 runs). The network is then trained on each shuffled set, yielding five slightly different trained networks with different starting weights and different training data. The result of this test is shown in Fig. 7: above  $3 - 5\sigma$  signal level the difference between the five models is only a few percent in both the precision and recall metrics, suggesting a robust model.

## 6. Post-processing and validation

The raw output of the network sometimes contained single false positive points and also smaller sections of flares that were not marked (false negatives), as seen e.g. in Fig. 3. These issues can be solved by smoothing the raw output by a median filter. The size of the filter, however, should be chosen carefully: a too short kernel size might be not efficient enough, while a too large kernel can filter out shorter events. Fig. 4 shows the test showing the effect of different kernel size on the precision/recall metrics. A window size of 47 and 63 points yielded the best results with a cutoff of 0.2. For the Kepler data we chose a 47 point-wide kernel, as it gave somewhat steadier output, while for the injected flares we used a 63 point filter after averaging the predictions of different K-Fold weights with a cutoff of 0.3 (see Sect. 7). When selecting the filtering method before validation we favored those options that yielded somewhat higher recall at the cost of losing some precision. Picking a kernel that filters out smaller events among false positives would give higher precision values, but these can be selected later at the validation phase.

The network managed to find the flare events correctly, but in several cases it seemed to neglect part of the eruption. To correct these, and to filter out possible remaining false positive detections, we decided to add a further post-processing step to mark



**Fig. 2.** Examples from the set used as ‘astrophysical noise’. Larger points indicate points predicted as possible flares by the trained network. From top to bottom: Algol type-,  $\beta$  Lyrae type eclipsing binary, RR Lyrae. The few-point false negative detections make post-processing and validation a necessary, but also a relatively easy task. The false positive detections in the plots will be cleaned by median filtering before the validation step.

these missing points, to detect possible false positives, and also to have an output that is more useful for physical analysis.

For each marked section, we took a part of the light curve with a length that equals three times the duration of the originally marked points, centered on the middle of the selected window (see Fig. 5. for an example). We fitted these points iteratively with a second-order polynomial filtering points out above  $0.5\sigma$ , then subtracted this trend. The residual is used to locate the flare event in the window. To eliminate outliers which can mislead the algorithm, we used a median filter with a width of 3 points. After this, the original light curve was fitted again with a combination of a second-order polynomial and the analytic flare model of Davenport (2016). If the time scale of the light curve variability is comparable to the length of the selected window, the second-order polynomial may not fit properly the background, thus a higher order is needed. To overcome this issue, we repeated the fitting procedure using a third-order polynomial and calculated the Bayesian Information Criterion (BIC; Liddle 2007) for both to select the better model. The BIC is defined as

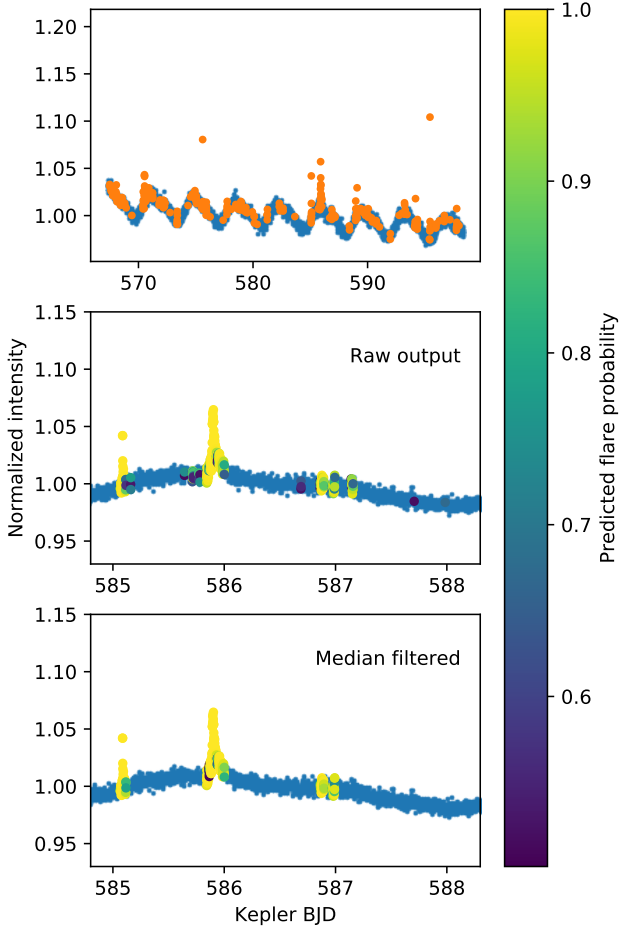
$$\text{BIC} = n \ln \left( \frac{\sum (y_i - \hat{y}_i)^2}{n} \right) + k \ln(n), \quad (1)$$

where  $n$  is the number of points,  $k$  is the degree of freedom, and  $y$  and  $\hat{y}$  are the measured and modeled flux values, respectively. The better the model the lower the BIC. The final model was removed from the light curve and the residual  $\sigma$  was used as the noise level to calculate the signal-to-noise ratio. If there still was at least three consecutive points above  $1.5\sigma$  noise level an additional fit is performed for complex events. This step was repeated until the highest point went down the noise level.

From the marked events only those can be considered as real flares with high probability that have their peak above  $5\sigma$  over the noise level. Below this, the rate of false positive events is increasing rapidly (see Sect. 7). The output also includes the beginning and end time of the events (those points where the fitted model reaches  $1\sigma$  level) and the equivalent duration of the flares (their integrated intensity over the event duration). Following the work of Kővári et al. (2007), the latter can be used to estimate the flare energy from the quiescent flux of the star assuming a simple black body model, which requires the knowledge of stellar effective temperature and radius. We collected these parameters from the Kepler Input Catalog (Brown et al. 2011) and calculated the flare frequency distributions (FFD) for some example stars that are shown in Fig. 6. The energy distribution characteristics of the recovered flare energies is similar to other methods, but a large number of low-energy events are recovered, which will be helpful in order to obtain information on a wide energy range.

## 7. Performance

We tested the performance of the network on two data sets: a set of Kepler light curves that were not used for training and validation steps, and light curves with injected flares. Both methods have their drawbacks: with real light curves we can compare the network performance to a human performing the same task, but we do not know the ground truth, i.e., where the flares occurred in reality, and also it is much harder to obtain manually flagged data as generate light curves. Artificial light curves are easy to generate, and we know precisely, where the flares are, but their characteristics might be somewhat different as real events.

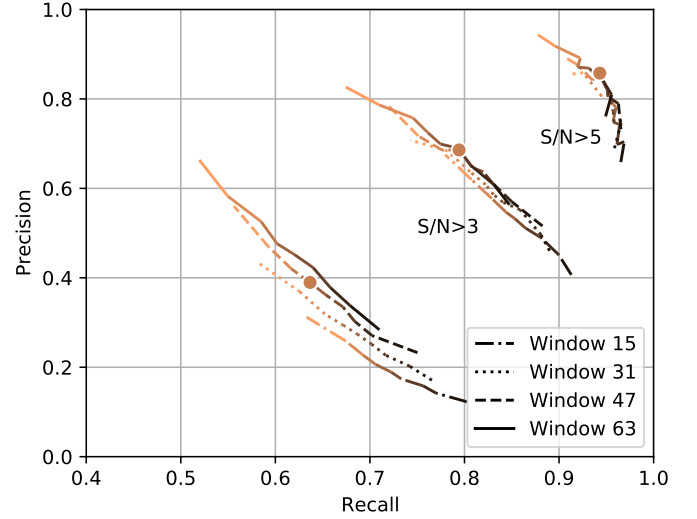


**Fig. 3.** Example output from the validation data set (KIC 012156549). The top panel shows the input light curve (blue points) with all the data points marked by the neural network as flares (orange points). The middle panel is zoomed in, color code showing the raw output from the predictions where the output probability higher than 0.5. There are a few points within flares that are not marked (false negatives), and some single points that are incorrectly identified as flares (false positives). This can be rectified by smoothing the output probabilities by a median filter (bottom plot panel).

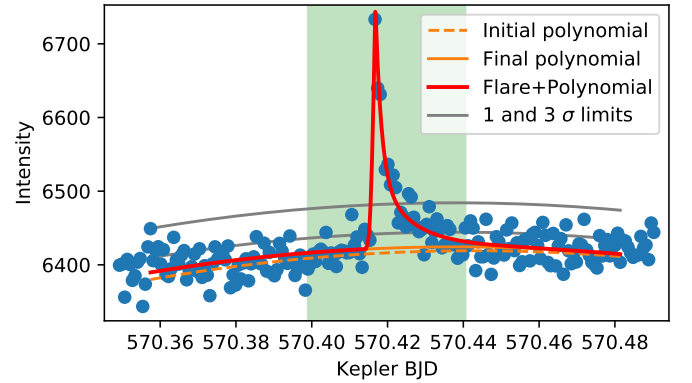
To generate the flare injected data, we selected ten stars from our sample, and removed the flares by smoothing, using multiple passes of a median filter. Then, we injected ten flares into each data set with peaks ranging between 2–15 times the noise level. Those points were marked as part of the event that were above the noise level of the light curve. These steps were repeated 100 times for each star, yielding 10 000 flares in  $\approx 30\,000$  days of data.

The performance of the network for the Kepler data set is shown in Fig. 7, where we plotted the precision and recall metrics as a function of the signal-to-noise ratio (left panel) and the measured relative amplitude of flares (right panel). The test shows that above  $5\sigma$  signal level the LSTM was able to recover  $\geq 80\%$  of the manually marked flares with a precision of  $\geq 70\%$ . The shape of the metric curves is different in the right plot. The low precision is caused by the very low number of events: above  $10^{-2}$  relative amplitude there were only a few dozen events.

For the injected data set, first, we tried prediction with the K-Fold weights. We found that, in contrast with the Kepler data set, using only a single K-Fold weight yielded a significantly worse



**Fig. 4.** Precision–recall curve with median filters of different kernel sizes. The three batches show all flare events from the test data (left), and cases with flares, in which the highest point compared to the noise in the light curve is above  $3\sigma$  (middle) and  $5\sigma$  (right). Color coding shows different cutoff levels above which a flux measurement is considered to be a flare. The darker the color the higher the cutoff threshold. The point marks the location of cutoff value 0.2 for 47-point-width median filtered case. See text for details.



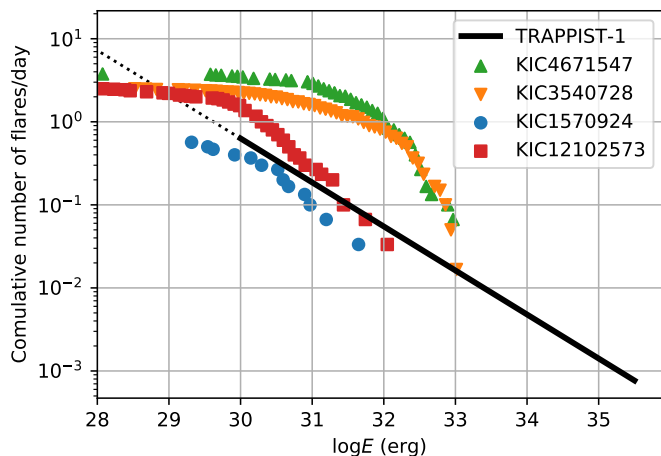
**Fig. 5.** An example for the validation. Green area shows the section marked as a probable flare. Additional flare-length windows before and after the marked region are selected, and fitted by a low-order polynomial. The fit is subtracted from the light curve, and the residual is fitted by another low-order polynomial and an analytical flare model. If the peak of the fit is above the  $3\sigma$  noise level, the event is marked as validated.

performance. Therefore, we tried to weighted average the predictions yielded by different K-Fold weights, which improved the results considerably. In this case, if one of the networks predicted a point to be a flare with a probability of  $> 0.98$ , we kept it as a flare event, regardless of the results of the other networks. The results of this test are shown in Fig. 8.

The typical runtime for one light curve with a length of one quarter was  $\approx 25$  seconds on a GPU with a batch number of  $b = 16$ .

As it can be seen the LSTM is performing better in finding light curves in less noisy areas than finding large amplitude flares in the noise. The non-zero recall number in case of the lowest S/N values is introduced by the flare fitting algorithm. First, it





**Fig. 6.** Four example flare frequency distributions of KIC 4671547, KIC 3540728, KIC 1570934 and KIC 12102573 (colored symbols). As a comparison we adopted and plotted the fit of FFD of the well-known star TRAPPIST-1 (black line) from Vida et al. (2017).

tries to find and fit the significant outlier points in a given flare region regardless of their origin, second, sometimes in the residual of a non-perfect fit, which is caused by the difference between the shape of the model and observed flare, some information remains that is considered to be a flare. Moreover, the number of flagged points within a flare is usually different in the manually and automatically labeled data set, i.e. the duration of flares are different, which gives the opportunity to find different number of flare-like events within a region of a given flare.

### 7.1. Evaluation on TESS sample

We tested the network also on TESS data, in a similar way as it was performed by Feinstein et al. (2020). Light curves were downloaded from MAST using the *Lightcurve* tool<sup>6</sup> (Lightcurve Collaboration et al. 2018). Labels for flares were obtained from Günther et al. (2020). This set contains a large amount of flaring light curves, but the labels are not perfect, there is a number of missing flares, or incorrect labels: these events will bias our comparison yielding incorrectly false positive or false negative detections. As the RNN requires equidistant sampling, the light curves were resampled to 2 minute cadence, which is equal to the TESS short-cadence sampling.

We predicted flares using all the weights saved during the Kepler K-fold test, and additionally we trained the network with all available Kepler light curves and utilized those weights as well. In the individual tests we were not able to recover as many flares as in case of the Kepler data. As it turned out, the main problem was the normalization of light curves. If we transform the flux values to the range of 0–1, the large amplitude flares suppress the small events making those impossible to identify. To overcome this problem, we sigma clipped the large flares before normalization. However this step raised another issue, the height of high amplitude flares went way above unity, which made the RNN uncertain, yielding much lower probabilities for these previously well recovered events. Thus, we somehow had to combine the results of different predictions. After an extensive test, we found that the best solution is to use three weights rather than only one, predict the location of flares in light curves nor-

malized by both methods and weighted average their predictions. The weights are unity, unless a given instance predicts an event with a probability larger than 0.9, then its prediction takes over the others. This way both small and large events are captured, and the false negative ratio is minimized.

The precision and recall values as a function of signal-to-noise ratio and relative flare amplitude are shown in Fig. 9. After a precision–recall test, similar to the one presented in Fig. 4, the raw predictions were median filtered with a window length of 32 and points were considered to be flares above probability threshold of 0.5. Although, the TESS data have different sampling and noise characteristics that may influence the results, our test showed that we are able to recover a similar number of flares as in case the Kepler data shown in Fig. 7–8. The shape of the precision curves are similar too, but the absolute values are 10–40% lower. Plotting the individual events marked as flares by our network, we found flares in the order of 4–5000 missing from the original catalog with similar characteristic to those of plotted in Fig. 10. Considering the number of flares in the catalog of Günther et al. (2020), which is about 8600, this difference is understandable. This test showed that even the current network is able to generalize and find flares – with similar effectiveness – in completely new data having previously unseen sampling and characteristics.

The *flatwrm2* code and the network used by Feinstein et al. (2020) use both different data sets and approaches. The latter is trained on TESS data that have different characteristics, and has a larger flare sample compared to the Kepler sample. Another important difference lies in the scaling and normalization of the input data: while we normalize the light curve as a whole, in case of *stella* each window is normalized separately. Therefore, *flatwrm2* might perform worse in cases where the light curve itself is peculiar, and the normalization process yields data with too large or too small events that are not recognized by the neural net. On the other hand, *flatwrm2* could outperform *stella* in the case of atypical flare shapes, complex flares or long flares, due to the network remembering its previous states thanks to the forget/update gates in LSTM. While *flatwrm2* was trained on Kepler short-cadence data only (and artificial light curves based on Kepler data), the LSTM network could be better in generalization. However, a qualitative comparison of the two methods is currently not possible given their mission-oriented training sets.

## 8. Failures

Research is rarely a linear process with forming the idea, working through it and concluding the results. This is especially true for developing machine learning methods – in this section we will shortly summarize those experiments, that did not work at all or gave inferior results.

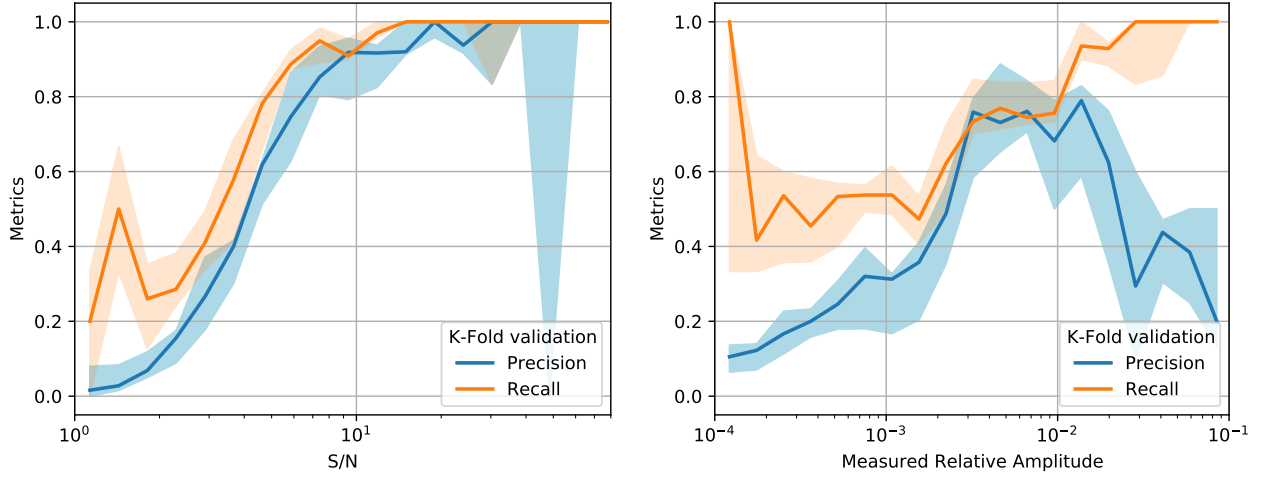
Initially we tried selecting the best candidates based on simple model data only, and use this weights on real data, but we found that over-simplified light curve models yield unusable model weights and architectures.

As mentioned in Sect. 2, standardizing input data is crucial. Here, simply dividing with the mean or median of each data set did not work, and also normalizing the data with their standard deviation proved problematic: in some cases larger flares were scaled up, and they got neglected by the network.

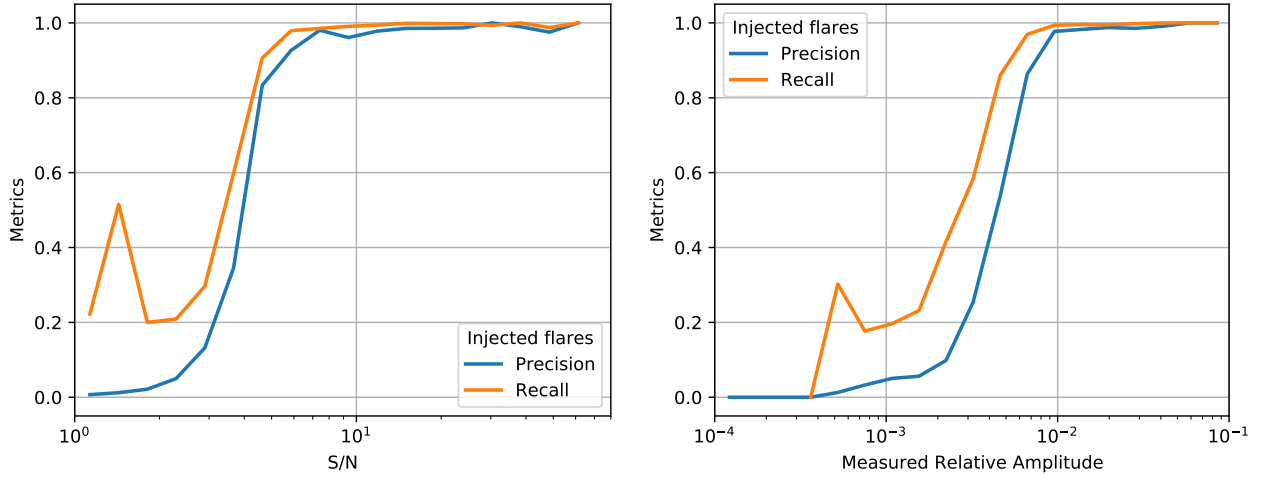
We experimented with a large number of network architectures that did not converge at all, or did not yield the expected results. We tried:

- simple dense networks with different sizes;

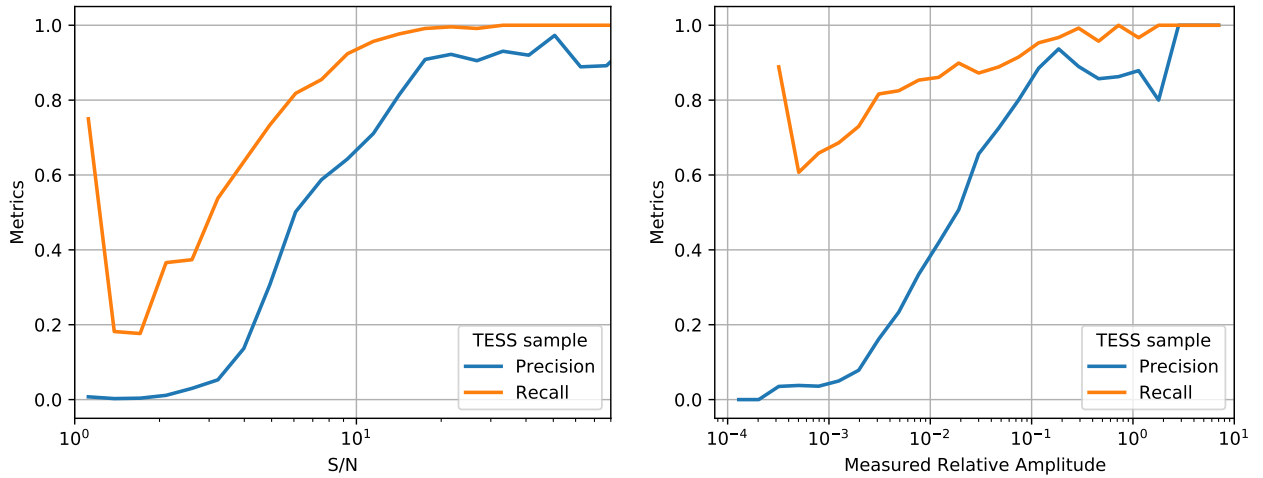
<sup>6</sup> <https://docs.lightcurve.org>



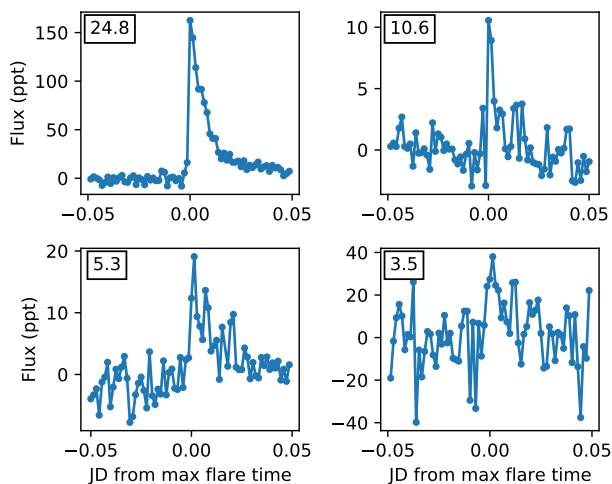
**Fig. 7.** The result of the K-fold cross-validation on the independent test data set. The curves show the precision (blue) and recall (orange) as a function of flare signal-to-noise ratio (left) and relative amplitude (right). The raw predictions were median filtered with a window length of 47. Points are considered to be flares above probability 0.2. Above the  $3 - 5\sigma$  signal level the uncertainty of the models as a result of different training sets is reduced to a few percent, suggesting that the trained model is robust, and that the training sets are sufficiently large. Note the logarithmic abscissa. The low precision in the right plot is caused by the very low number of events.



**Fig. 8.** Injection-recovery test using the best performing network, combining the weights of all the K-Fold results. The combined raw predictions were post-processed in the same way as in case Fig. 7. Note the logarithmic abscissa.



**Fig. 9.** The result of predicting flares in the TESS sample of Günther et al. (2020). These results were reached using the combined weights of different train runs on the Kepler sample. The raw predictions were median filtered with a window length of 31. Points are considered to be flares above probability 0.5. Note the logarithmic abscissa.



**Fig. 10.** Example high signal-to-noise ratio false positive flares found by our LSTM algorithm in the TESS sample of Günther et al. (2020). The numbers in the upper left corners show the S/N ratios. Less significant detections are dominated by non-flare events, i.e. genuine false positives.

- convolutional networks;
- hybrid networks utilizing both recurrent nets and dense or convolutional networks in parallel

with various inferior results (our takeaway lesson was that a more complex network is not necessarily performing better). Adding an extra dense layer with 32 units after the selected recurrent layers did also not improve the results. Bidirectional recurrent networks (that analyze the input data in both direction) worked in some cases slightly better, but at the cost of doubling the runtime, therefore we decided to use one-directional analysis (with this the prediction time for a single-quarter Kepler light curve is  $\approx 13$  seconds on GPU with a batch number of  $b = 64$ ).

Changing the window size for training did not improved the results either. Using a shorter window, the number of false positive detections increased significantly, while larger windows tend to smear out smaller events.

## 9. Code availability

To make our recurrent neural network a useful tool for finding flares in the Big Data era, we prepared a user-friendly code, dubbed as `flatwrm2`, which is available through a GitHub repository<sup>7</sup>, and can be installed via PyPI. Passing a flux time-series, `flatwrm2` will prepare the light curve, predict and validate the possible flare events. The user will encounter a bunch of output parameters, such as the location, FWHM, relative amplitude and signal-to-noise ratio of flares, of which the latter can be used to select the more likely intrinsic events. Besides the raw code, we have written a tutorial Notebook which guides the user through the prediction and validation steps.

## 10. Summary

We presented an experiment to detect flares in space-borne photometric data using deep neural networks (NNs). After testing a large number of network architectures, we concluded that the best performing network was a recurrent neural network based

on using Long Short-Term Memory (LSTM) layers. After training a network using a set of artificial light curves, and real short-cadence Kepler observations of flaring and non-flaring stars, the network was not only able to detect flares with peaks over  $5\sigma$  with 80–90% recall and precision, but also was capable to distinguish false signals that typically confuse flare-finding algorithms (e.g. maxima of RR Lyr stars) from real flares.

In the future we plan to analyze a large number of Kepler and TESS data to gain a wider knowledge of flaring stars. The code with the weight files are available in the public domain for the scientific community.

## 11. Software

Python (Van Rossum & Drake 2009) –Numpy (van der Walt et al. 2011) –Pandas (McKinney 2010) –Scikit-learn (Pedregosa et al. 2011) –Tensorflow (Abadi et al. 2015) –Keras (Chollet & others 2018) –Matplotlib (Hunter 2007) –scipy (Virtanen et al. 2020) –PyMacula (Kipping 2012) –Lightkurve (Lightkurve Collaboration et al. 2018)

**Acknowledgements.** The authors would like to thank deeplearning.ai and Coursera for hosting their online course on different aspects on machine learning, without which this work would never be possible. We thank the referee for their swift replies and the valuable remarks that considerably helped us to improved the manuscript and the code availability. BS was supported by the ÚNKP-19-3 New National Excellence Program of the Ministry for Innovation and Technology. This project has been supported by the Lendület Program of the Hungarian Academy of Sciences, project No. LP2018-7/2020, the NKFI KH-130526, NKFI K-131508 and 2019-2.1.11-TÉT-2019-00056 grants. On behalf of "Analysis of space-borne photometric data" project we thank for the usage of ELKH Cloud (<https://science-cloud.hu>) that helped us achieving the results published in this paper. Authors acknowledge the financial support of the Austrian-Hungarian Action Foundation (95 öu3, 98öu5, 101öu13).

## References

- Abadi, M., Agarwal, A., Barham, P., et al. 2015, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, software available from tensorflow.org
- Arras, L., Osman, A., Müller, K.-R., & Samek, W. 2019, arXiv e-prints, arXiv:1904.11829
- Borucki, W. J., Koch, D., Basri, G., et al. 2010, Science, 327, 977
- Brown, T. M., Latham, D. W., Everett, M. E., & Esquerdo, G. A. 2011, AJ, 142, 112
- Cho, K., Van Merriënboer, B., Gulcehre, C., et al. 2014, arXiv preprint arXiv:1406.1078
- Chollet, F. & others. 2018, Keras: The Python Deep Learning library
- Davenport, J. R. A. 2016, ApJ, 829, 23
- Dozat, T. 2015, [http://cs229.stanford.edu/proj2015/054\\_report.pdf](http://cs229.stanford.edu/proj2015/054_report.pdf)
- Feinstein, A. D., Montet, B. T., Ansdell, M., et al. 2020, AJ, 160, 219
- Günther, M. N., Zhan, Z., Seager, S., et al. 2020, AJ, 159, 60
- Hochreiter, S. & Schmidhuber, J. 1997, Neural computation, 9, 1735
- Hunter, J. D. 2007, Computing in Science & Engineering, 9, 90
- Khodachenko, M. L., Ribas, I., Lammer, H., et al. 2007, Astrobiology, 7, 167
- Kővári, Zs., Vilardell, F., Ribas, I., et al. 2007, Astronomische Nachrichten, 328, 904
- Kingma, D. P. & Ba, J. 2014, arXiv preprint arXiv:1412.6980
- Kipping, D. M. 2012, macula: Rotational modulations in the photometry of spotted stars
- LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. 1999, in Feature Grouping, ed. D. Forsyth (Springer)
- Liddle, A. R. 2007, MNRAS, 377, L74
- Lightkurve Collaboration, Cardoso, J. V. d. M., Hedges, C., et al. 2018, Lightkurve: Kepler and TESS time series analysis in Python, Astrophysics Source Code Library
- McKinney, W. 2010, in Proceedings of the 9th Python in Science Conference, ed. S. van der Walt & J. Millman, 51 – 56
- Oh, S. L., Ng, E. Y., San Tan, R., & Acharya, U. R. 2018, Computers in biology and medicine, 102, 278
- Oláh, K., Kővári, Zs., Günther, M. N., et al. 2021, A&A, 647, A62

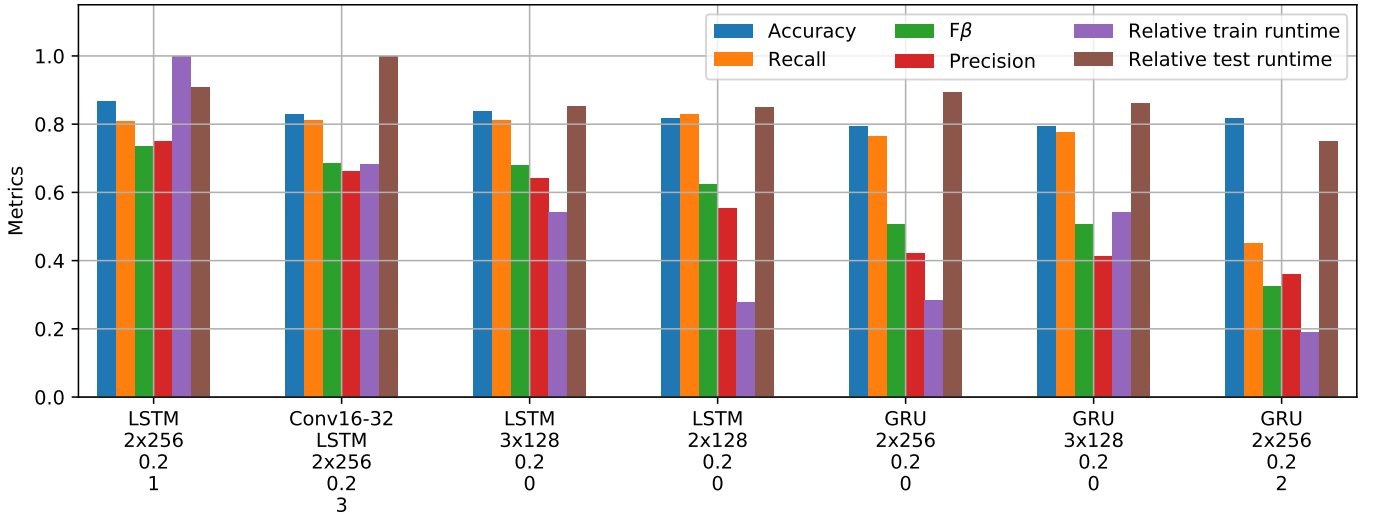
<sup>7</sup> <https://github.com/vidakris/flatwrm2>



- Pedregosa, F., Varoquaux, G., Gramfort, A., et al. 2011, *Journal of Machine Learning Research*
- Rauer, H., Catala, C., Aerts, C., et al. 2014, *Experimental Astronomy*, 38, 249
- Ricker, G. R., Winn, J. N., Vanderspek, R., et al. 2015, *Journal of Astronomical Telescopes, Instruments, and Systems*, 1, 014003
- Roettenbacher, R. M. & Vida, K. 2018, *ApJ*, 868, 3
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. 2014, *The journal of machine learning research*, 15, 1929
- Stelzer, B., Damasso, M., Scholz, A., & Matt, S. P. 2016, *MNRAS*, 463, 1844
- Sutskever, I., Vinyals, O., & Le, Q. V. 2014, *arXiv e-prints*, arXiv:1409.3215
- van der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, *Computing in Science and Engineering*, 13, 22
- Van Rossum, G. & Drake, F. L. 2009, *Python 3 Reference Manual* (Scotts Valley, CA: CreateSpace)
- Vida, K., Kővári, Zs., Pál, A., Oláh, K., & Kriskovics, L. 2017, *ApJ*, 841, 124
- Vida, K. & Roettenbacher, R. M. 2018, *A&A*, 616, A163
- Virtanen, P., Gommers, R., Oliphant, T. E., et al. 2020, *Nature Methods*, 17, 261
- Walkowicz, L. M., Basri, G., Batalha, N., et al. 2011, *AJ*, 141, 50
- Yelle, R., Lammer, H., & Ip, W.-H. 2008, *Space Sci. Rev.*, 139, 437

## Appendix A: Performance of different architectures

In Fig. A.1, we show the performance of the RNN architectures we designed, trained and tested on the same data set. As a single metric is not enough to select the best one, we plotted the accuracy, recall, precision,  $F\beta$ , along with the training and evaluation time compared to the slowest instance, respectively. The TP, FP and FN values were calculated from the maxima times of detected flares. The accuracy is biased as the number of TN flares, considering all the points that are not flagged, are orders of magnitude larger than the flagged points. The rows in labels are the name, units and layers of the kernels, the dropout value and number of the trained dense layers after the RNN units. The last dense layer that is used to link the outputs is not shown. In case of the second network, we used two CNN layers with filter sizes of 16 and 32 before the RNN. We selected the third architecture to work with, as it yielded almost the same results as the first two, but with a reduced training time of only 40% and 20%.



**Fig. A.1.** The performance of the designed and tested RNN architectures. The bars show the accuracy (blue), recall (orange),  $F\beta$  (green), precision (red) and the relative runtimes of training (pink) and evaluating (brown) compared to the slowest instance, respectively. The networks that loss started to increase dramatically after some epoch were removed.