

Machine Learning for Solar Flare Detection

Raymond Blaha, Paul Hwang, David Nezelek

Practical Data Science

December 22, 2023

Introduction

The research task, given to us by astrophysicist Vinay Kashyap of the Center for Astrophysics (Harvard & Smithsonian) was to detect a solar flare, locate it within a known active region on the surface of the sun, and measure its properties. Solar flares are intense bursts of radiation coming from the release of magnetic energy. Flares are associated with coronal mass ejections, large expulsions of plasma which can interfere with radio and satellite communications and even electrical power transmission.

NASA creates a historical record of flares using a computational algorithm. This algorithm is known to be insufficiently sensitive to smaller flares. For our project, we used machine learning models to detect solar flares, then tested our findings against the historical record. We hoped to not only find flares in the record, but to improve upon the record by detecting flares not picked up by the algorithm.

Data

Sources

The GOES solar data and solar flare data were obtained from National Oceanic and Atmospheric Administration (<https://www.ngdc.noaa.gov/stp/satellite/goes-r.html>) for different satellites (from GOES 13 to GOES 18).

Preprocessing: munging, merging

Reading in the data (in ncdf4 format) in R, we extracted the solar flux and time from solar data and occurrence of solar flare peaks from solar flares data for each satellite. Then, we merged the datasets so that for each satellite, we had information about time, fluxes, and presence of solar flare. Afterwards, we merged the datasets for all of the satellites, prioritizing the later satellites over the earlier satellites, i.e., if there are values from different satellites for a specified time, GOES-18 was prioritized, followed by GOES-17 and so forth, to yield a single flux value for the specified time.


Models

Time Series

We used two different machine learning algorithms to run time series analysis: Long-Short Term Memory (LSTM) and One-Dimensional Convolutional Neural Network (1D-CNN). To use these models, we first reformatted the data, as these time series models require a 3D data structure, while the structure of the time series data is in 2D. We grabbed 100 data points and flattened them, putting in 100 data values of each flux as an input for the 50th minute. The simplified version of this procedure, using a span of three data points, is shown below with Figure 1. We also scaled the data from 0 to 1, as our data spanned from 10^{-8} to 10^{-4} , since the algorithm handles values better when they are not too small or too large.

Figure 1: Representation of data reshaping

	Xrsa_flux	Xrsb_flux	Flare
Day1	1	2	0
Day2	3	4	1
Day3	5	6	0
Day4	7	8	1



	Xrsa_flux	Xrsb_flux	Flare
Day2	(1,3,5)	(2,4,6)	1
Day3	(3,5,7)	(4,6,8)	0

From there, we used different model architectures, including different builds of LSTM, 1D-CNN, Bidirectional elements (which look at the architecture in both directions in time), and combinations of all of the above. We found that the general performance of 1D-CNN was better in terms of accuracy, recall, and computation time.

Autoencoder

With the initial promise of the 1D-CNN models, we recognized the need for a model that could leverage one-dimensional convolutions and learn autonomously. We determined that an unsupervised learning model would be the best fit for the requirements of the project.

Data Preprocessing:

1. Data loading and Time Conversions:

- The solar flux data is read in as a Pandas DataFrame.
- This format is ideal for data manipulation and analysis.
- The 'time' column is converted to a daytime object, and timezone information is removed to standardize the timestamps. This step is important for a consistent time series analysis.

2. Normalization with MinMaxScaler:

- The key attributes of the data are the 'xrsa_flux' and 'xrsb_flux' columns in the Pandas DataFrame. These columns were normalized using the MinMaxScaler.

3. Data Splitting:

- a. The dataset is split into training, validation, and testing sets, at ratios of 60%, 20%, and 20%, respectively. These ratios allow for a robust and diverse dataset on which the model can be trained.

4. Saving Training, Validation, and Testing datasets:

- a. The training, validation, and testing dataset after preprocessing are saved as individual CSV files to facilitate reading the data into different training and testing scripts, and to minimize computational complexity.

The Autoencoder we developed consists of two primary components, the encoder and the decoder.

Encoder (Figure 2.1):

The encoder takes the input sequence of the XRSA and XRSB flux data, which is structured as a two-dimensional array with a shape of (SEQUENCE_LENGTH, len(norm_cols)), and compresses the data into a lower-dimensional representation. This is possible through a series of convolutional and max pooling layers, which help in extracting the important features from the data, followed by a dense layer that produces the latent space representation.

1. Conv1D Layers:

- a. The first Conv1D layer has 16 filters, a kernel size of 3, and utilizes “ReLU” activation. Ensuring that the model is able to capture local patterns in the XSRSA AND XRSB solar flare flux data. The “same” padding is used to ensure that the output size is the same as the input size.
- b. The second Conv1D later, also utilizes “ReLU” activation and increases the filter count to 32. This layer in particular refines the feature extraction process by capturing more complex features in the data.

2. MaxPooling1D Layers:

- a. Following each instance of the Conv1D layer, a MaxPooling1D layer with a pool size of 2 is used. The importance of this layer is to ensure that the dimensionality is reduced allowing for features to be visible to the model and computational costs.

3. Dropout Layers:

- a. A dropout layer of 0.3 is selected after the MaxPooling1D. The dropout layer ensures that the model does not fall victim to overfitting. This is conducted by randomly setting a fraction of the input units to 0 at each update during training.

4. Flatten Layers:

- a. The Flatten layer converts the 2D feature maps into a 1D feature vector. This is important for the transition from convolutional layers to dense layers.

5. Dense Layers (Latent Space Representation)

- a. A Dense layer with the latent_dim units of 10 and “ReLU” activation. The units in this particular case represent the 10-dimensional latent space vector that contains the salient features of the input data. The 10-dimensional vector was chosen in this application to be the ideal balance between simplicity and sufficient complexity to represent the features while ensuring the model does not overfit.

Decoder (Figure 2.2):

The decoder then takes the condensed representation and attempts to reconstruct the original input data. It uses a series of transposed convolutional or rather deconvolutional layers, which work to upsample the data back to the original representation.

Decoder:**1. Dense and Reshape Layers:**

- a. The decoder starts with the Dense layer sized to upscale the latent representation, followed by the Reshape layer to prepare for transposed convolutional layers.

2. BatchNormalization Layer:

- a. This layer normalizes the activations of the previous layers, which improves the stability and speed of the model's training.

3. Conv1DTranspose Layers:

- a. This reverses the operations of the Conv1D, upsampling back to the original dimension. The first instance of the Conv1DTranspose layer contains 32 filters and the second Conv1D has 16 filters which both employ "ReLU" activation and "same" padding.

4. Final Conv1D Layer:

- a. The final Conv1D layer with a linear activation is used to reconstruct the original input dimensions allowing for the reconstruction of the solar flux data.

The key feature of the autoencoder model is its ability to reduce data into the most salient features, represented in a latent space. The autoencoder model works to identify and reconstruct the most significant patterns in the data, in this case solar flare peaks.

Figure 2.1: Encoder architecture

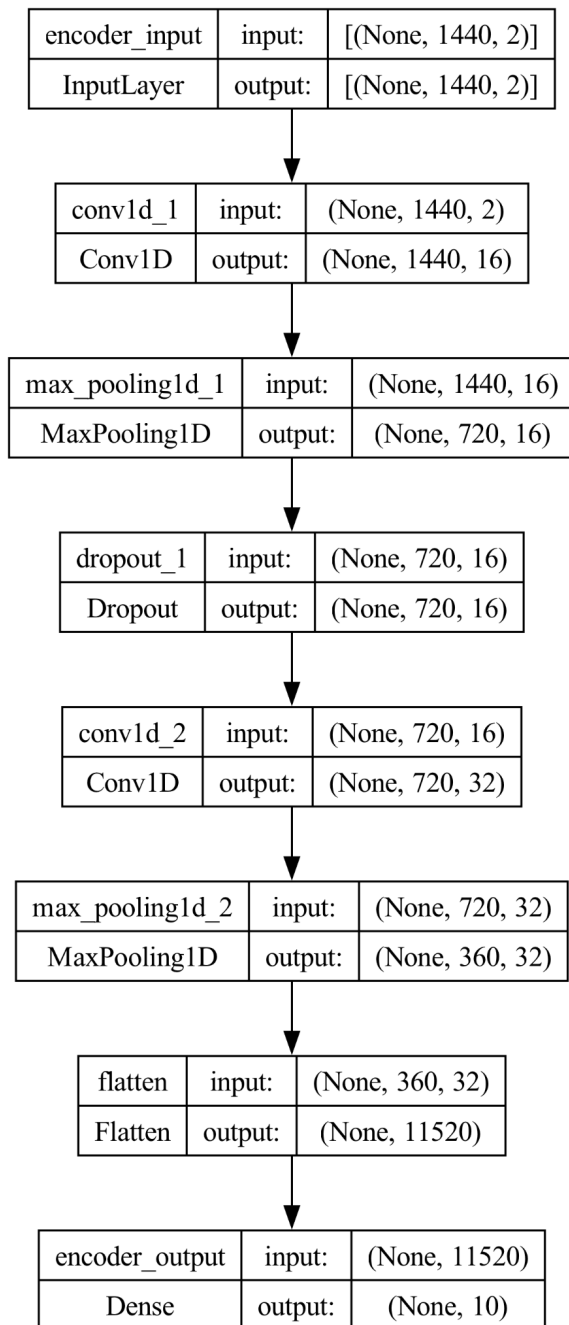
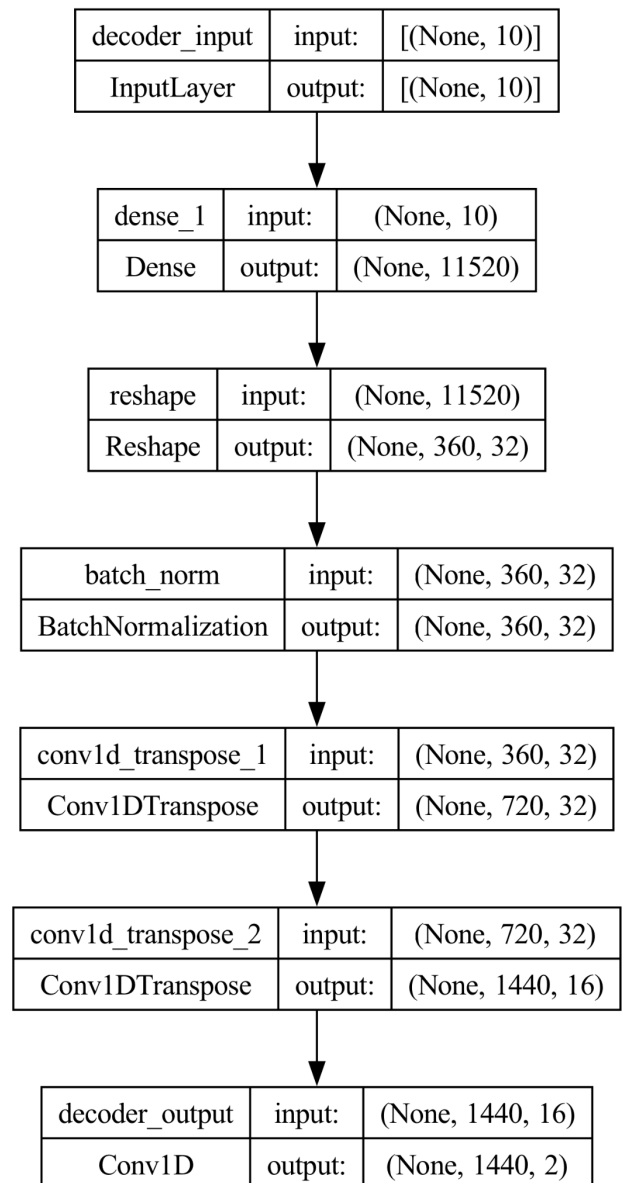


Figure 2.2: Decoder architecture



The autoencoder's effectiveness in detecting solar flares from the data, hinges on the model's ability to reconstruct the input that was fed into the encoder, all while emphasizing the anomalies that are the solar flare peaks. To maximize the model's ability to accurately detect solar flares, a custom loss function was designed to model more heavily for any failing to identify actual solar flares events, which are known in this application, as a false negative in our model's predictions.

Custom Loss Function:

When formulating the custom loss function, the end goal was to drastically underscore the false negatives within our model's predictions so that it guided us past the depreciation of any reconstructive errors. This will then allow the model to focus on higher levels of certainty when detecting the number of occurrences of solar flares. The team's approach to this, has enhanced the model's accuracy with identifying factual solar flare events. From this evidence, we can show that the autoencoder continues to help classify our most critical key piece of evidence within our dataset: finding the most dependable and accurate detection of solar flare anomalies.

The custom loss function can be described as:

$$Custom\ Loss(y_{true}, y_{pred}) = MSE(y_{true}, y_{pred}) + 2 \times FN(y_{true}, y_{pred}) + 0.5 \times FP(y_{true}, y_{pred})$$

Where:

$MSE(y_{true}, y_{pred})$ is the Mean Square Error as defined as:

$$MSE(y_{true}, y_{pred}) = \frac{1}{n} \sum_{i=1}^n (y_{true,i} - y_{pred,i})^2$$

Here, n is the number of samples, $y_{true,i}$ is the true value, and $y_{pred,i}$ is the predicted value for the i -th sample.

Where False Negatives (FN):

$FN(y_{true}, y_{pred})$ represents the loss from false negatives, calculated as:

$$FN(y_{true}, y_{pred}) = \frac{1}{n} \sum_{i=1}^n (y_{true,i} - y_{pred,i})^2 \times y_{true,i}$$

This term sums the squared errors but only for the instances where a solar flare is actually present (i.e., $y_{true,i} > 0$).

Where the False Positives (FP):

$FP(y_{true}, y_{pred})$ denotes the loss from the false positives, calculated as:

$$FP(y_{true}, y_{pred}) = \frac{1}{n} \sum_{i=1}^n (y_{true,i} - y_{pred,i})^2 \times (1 - y_{true,i})$$

Similar to the FN term, but in this instance, we are focused on the instances where the model predicts the solar flare in where there is no actual solar flare event.

Finally, the coefficients 2 and 0.5 are weighting factors which emphasize the importance of minimizing false negatives more than false positives. Essentially, this custom loss function will penalize false negatives more heavily comparative to false positives.

Anomaly Detection:

The main task of our autoencoder is to identify anomalies: the peaks of the solar flares. Given that solar flares typically are expressed as deviations from normal solar flux patterns, we can signal the occurrence of solar flares by identifying unusual patterns in our solar flux data.

One important distinction between our Autoencoder and a typical Autoencoder is that our model calculates the loss between the decoder and encoder using our custom loss function mentioned above. Since the custom loss function is designed to emphasize false negatives and false positives differently, it inherently affects how the model perceives and reconstructs the input data.

1. Reconstruction Error as Anomaly Indicator:

- a. After the model is trained, the model utilizes the custom loss function to form a viable anomaly detection system. In other words, due to the atypical nature of the peaks, we would expect a higher reconstruction error.
- b. The divergence in the reconstruction error influences how the model learns, being particularly sensitive to peaks of the solar flares.

2. Setting the Anomaly Threshold:

- a. We can set a threshold for flagging anomalies. This threshold is set at the 95th percentile of the reconstruction errors across the dataset. The 95th percentile was determined by modeling the tuning, which handled the variability and uncertainty in solar flare data. This provided a more robust model for anomaly detection.

Results

Metrics

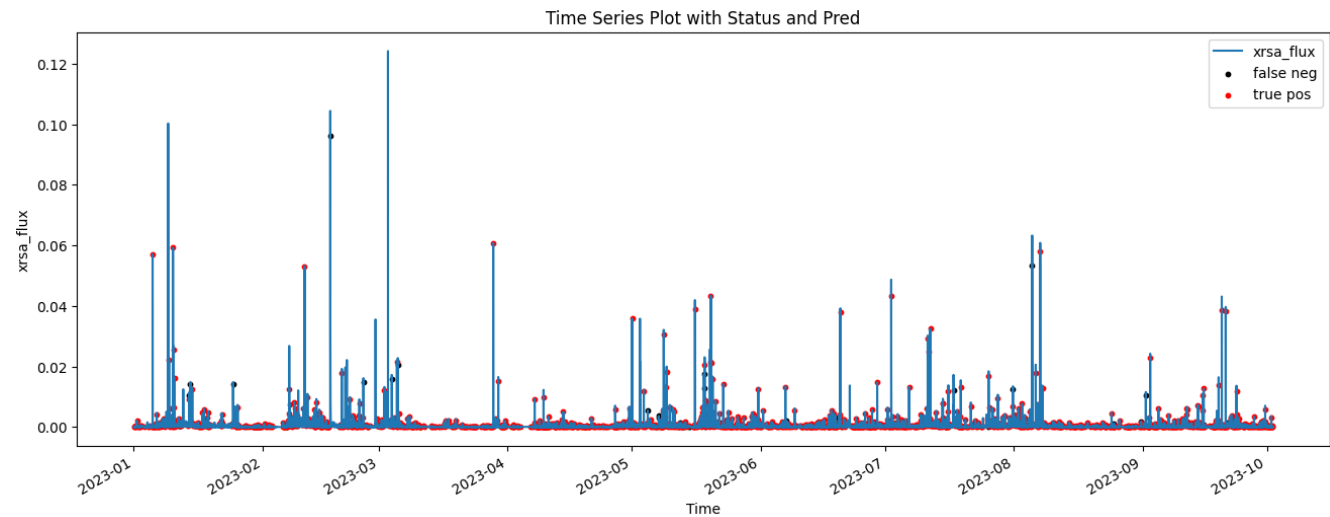
Any models that had LSTM structure (both simple LSTM and Bidirectional LSTM) had poor performance in precision and recall and took much longer periods of time, but 1D-CNN performed much better with those two metrics and took much shorter time.

Regarding the final architecture of 1D-CNN, we used two convolutional layers, a flattening layer, and two dense layers followed by a final dense layer. As can be seen from Figure 3, It has a strong recall for both flares and non-flares, but it has poor precision for flares as the model highlights the regions of potential flares. We can observe the general behavior in Figure 4 for the year of 2023.

Figure 3: Summary of CNN model performance

Classification Report:					
		precision	recall	f1-score	support
	0.0	1.00	0.92	0.96	388433
	1.0	0.07	0.93	0.12	2300
	accuracy			0.92	390733
	macro avg	0.53	0.93	0.54	390733
	weighted avg	0.99	0.92	0.95	390733

Figure 4: Visualization of ground-truth flares found and missed



We evaluated the Autoencoder with a multifaceted approach that includes the classification report, confusion matrix highlights, and ROC curve.

Classification Report:

The classification report boasts an impressive precision and recall for the majority class (0) - (non-instance of solar flare). Whereas, the results for the minority class (1) - (instance of solar flare) reflect that the model struggles to identify solar flares.

Precision and Recall of Anomaly Detection (Class 0):

1. With a precision, recall, and f1-score of 1.00, this indicates that the model can identify instances of non-peaks of solar flares every time and with high accuracy.

Precision of the Anomaly Detection (Class 1):

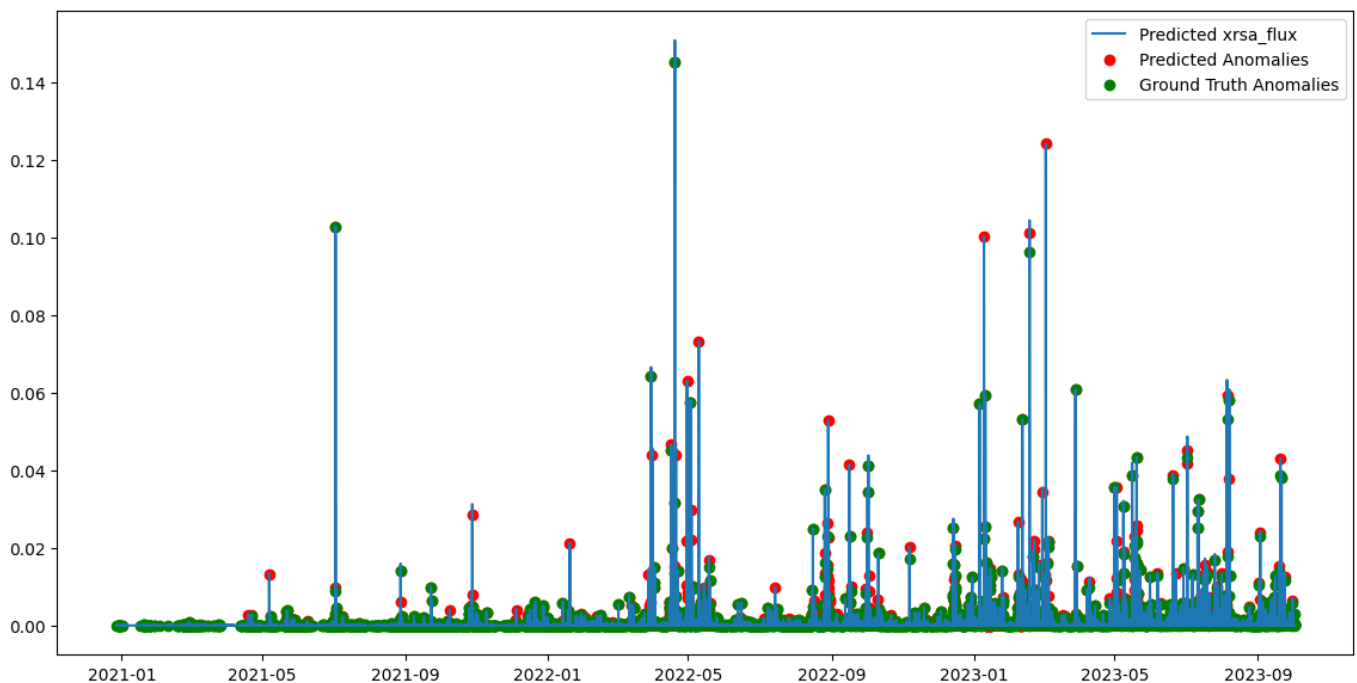
1. With a precision of 0.30 for the anomaly class, it suggests that when the model predicts the solar flare event, it is correct 30% of the time. Whereas, 70% of the remaining time may be false positives.

Recall of the Anomaly Detection (Class 1):

1. With a recall of .29, this suggests that the model captures 29% of actual solar flare events. This shows that there is a significant number of missed detections.

Given the slight improvement, it would be important to see how far the model predictions are to the ground truth. Refer to Figure 5, to see the difference in the model predictions to the ground truth being the current NASA algorithm for detection of solar flares.

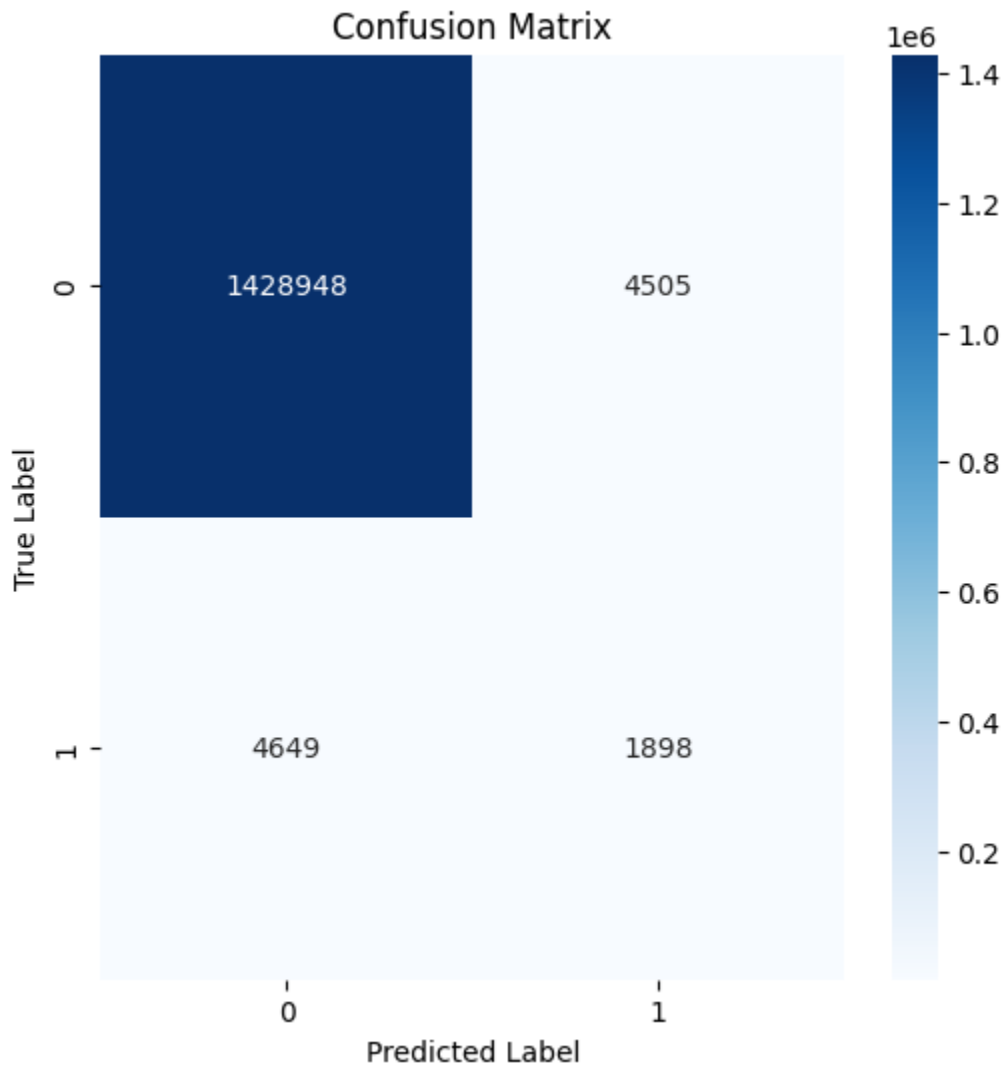
Figure 5: Visualization of flares predicted against ground truth



Confusion Matrix:

1. **True Negatives:** The model predicted the non-anomalous class 1,430,042 times.
2. **False Positives:** The model incorrectly predicted the anomalous class 3,411 times when it was actually a non-anomalous class.
3. **False Negatives:** The model incorrectly predicted the non-anomalous class 4,969 times when it was actually anomalous.
4. **True Positives:** The model correctly predicted the anomalous class 1,578 times.

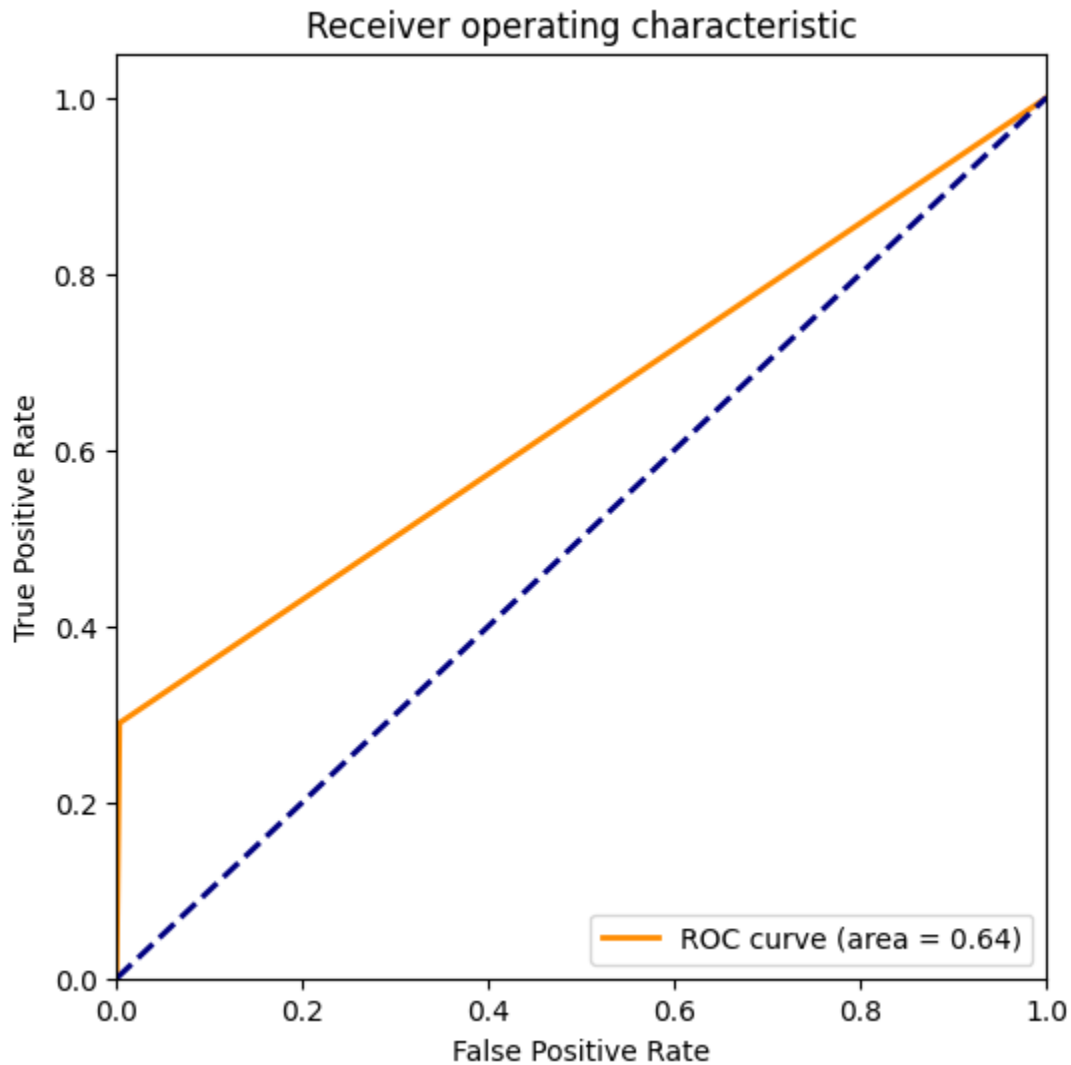
Figure 6: Confusion matrix



ROC curve:

With the area under the curve (AUC) of 0.64, the model indicates that it is struggling with the discriminative ability between non-anomalous and anomalous classes. The result of 0.64 suggests that this model is marginally superior to guessing between the non-anomalous class and anomalous class. Refer to Figure 7 for the ROC curve with the area under the curve.

Figure 7: ROC curve

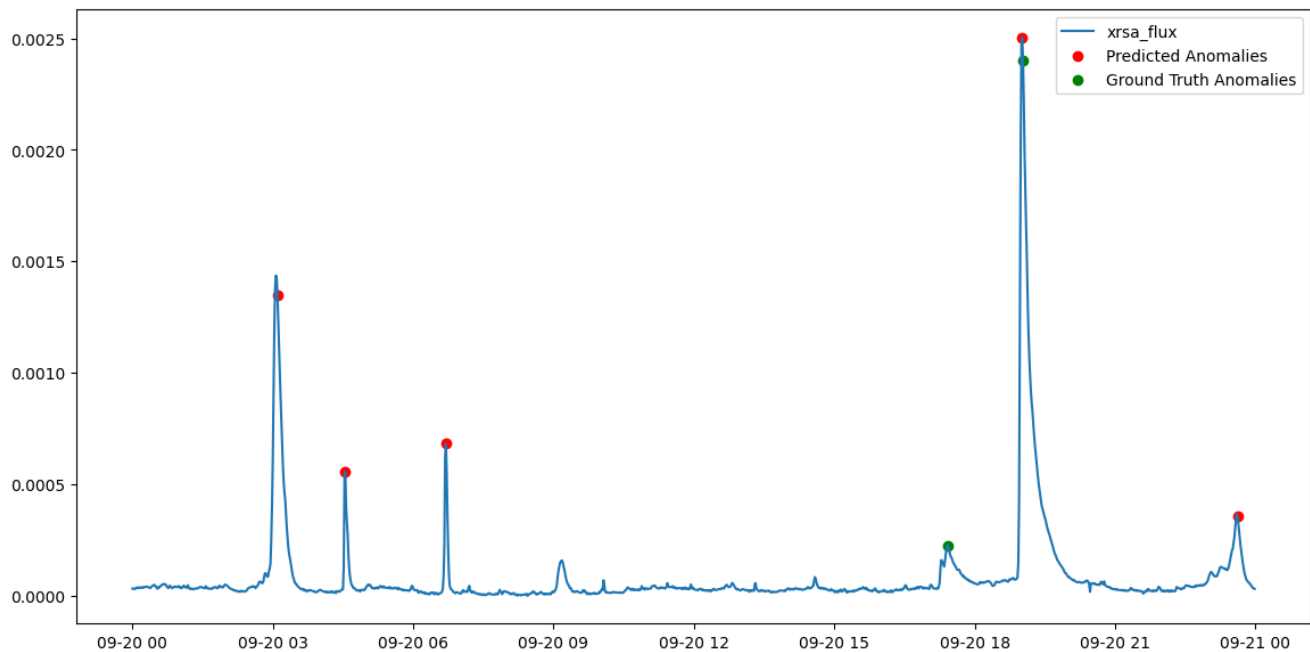


Analyzing a flare

Since the model trained on a 24 hour window of solar flux, we analyzed one 24-hour window to gain a better understanding of the model characteristics. Refer to Figure 8 for the 24-hour window of 09/20/2022, which we used to launch our exploration into comparing our model with the NASA algorithm.

Exploring the 24-hour window, it appears that the model is able to most accurately find solar flares or which are identified by the NASA algorithm at the true peak of the flux graph.

Figure 8: Flares on 9/20/22



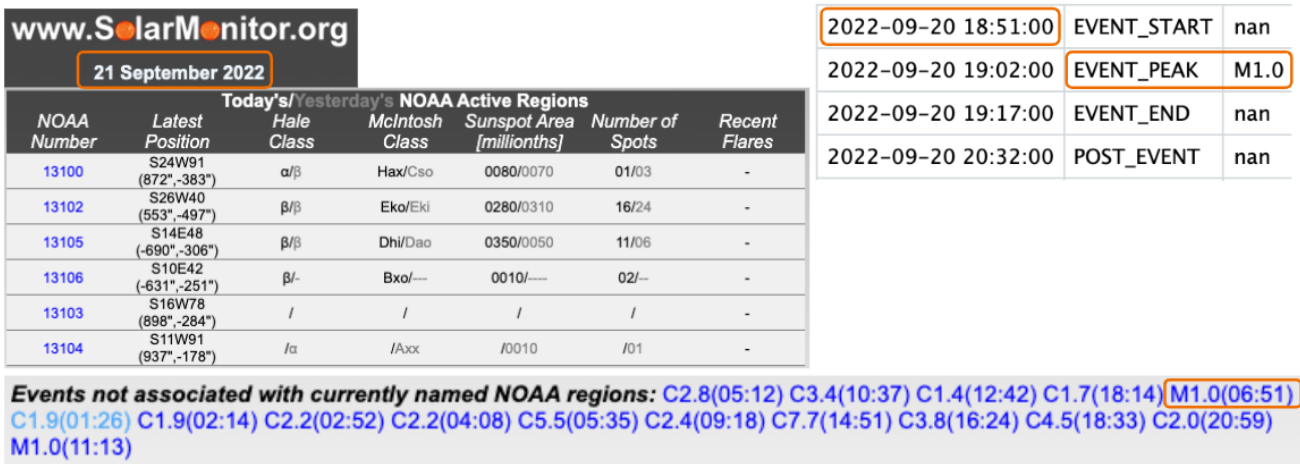
We selected the flare which appears in Figure 8 as a ground truth anomaly at 19:02 on 9/20/22. Our model found this anomaly at 7:01, which more precisely captures the peak intensity. Due to a date decoding error in our data preprocessing, the times (both predicted and ground truth) on the graph above are 12 hours before the true time. The error is evident when comparing the first cell in the table below (Figure 9), the erroneously preprocessed time, and the last cell in the top row, the flare ID from the original data.

Figure 9: Evidence of date decoding error

2022-09-20 18:51:00	1.758357e-06	EVENT_START	nan	202209210651
2022-09-20 19:02:00	1.013076e-05	EVENT_PEAK	M1.0	202209210651
2022-09-20 19:17:00	5.665353e-06	EVENT_END	nan	202209210651

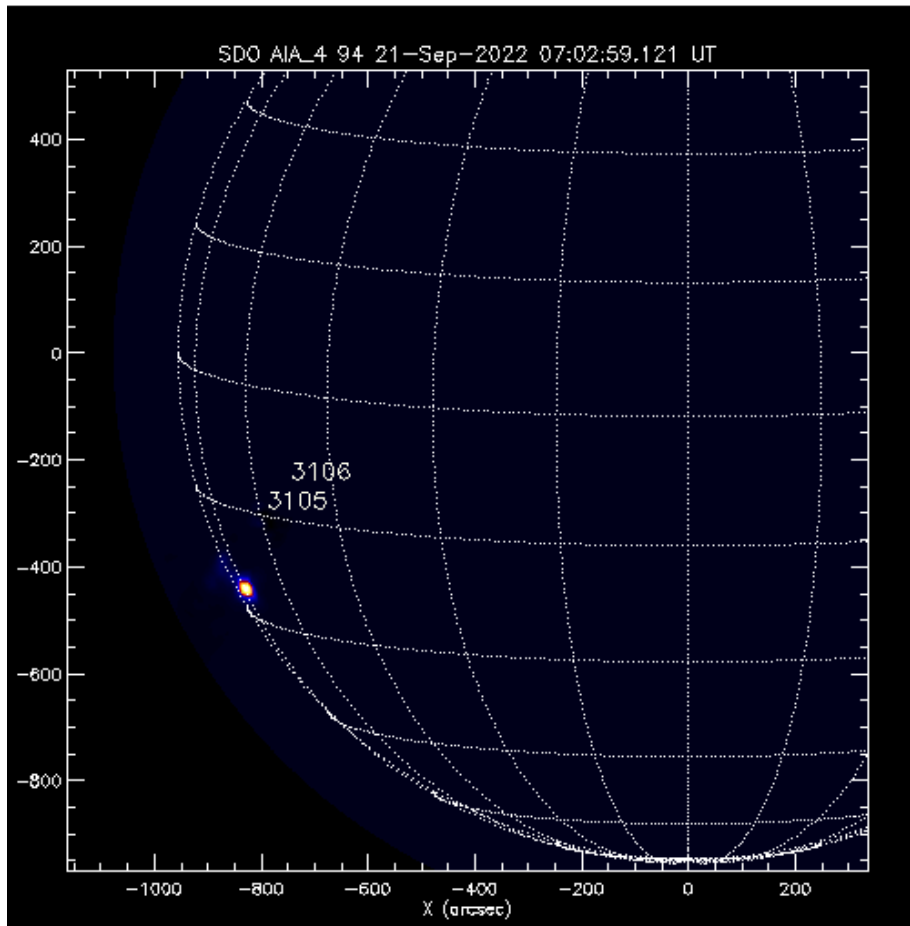
Since this error took place before any modeling, the discrepancy only becomes apparent when comparing the predictions with the historical record. This flare was found by our algorithm and cross-referenced with the historical record on SolarMonitor.org (Figure 10), maintained by the Solar Physics Group of Trinity College Dublin and the Dublin Institute for Advanced Studies. We found the start time for our flare at 6:51 on 9/21/22.

Figure 10: Flare record



As seen in Figure 10, the flare occurred in an active region which had not yet been named. To determine the active region, we then searched for an image of the location of the flare. We found it (Figure 11) on Imsal.com, a site hosted by Lockheed Martin Solar & Astrophysics Laboratory.

Figure 11: Flare location visualization




From the same page, we found what appeared to be the named region for the flare, in the last entry in the row: 3107 (Figure 12).

Figure 12: Flare active region assignment

Event#	EName	Start	Stop	Peak	GOES Class	Derived Position
2	gev_20220921_0651	2022/09/21 06:51:00	07:18:43	07:02:00	M1.0	S25E73 (3107)

As seen in Figure 10 above, the proximate active regions are five digit numbers beginning with 1. Searching the Heliophysics Event Catalog maintained by the Mullard Space Science Laboratory at University College London (helio.mssl.ac.uk), we confirmed (Figure 13) that active region 13107 was established on the following day.

Figure 13: History of active region



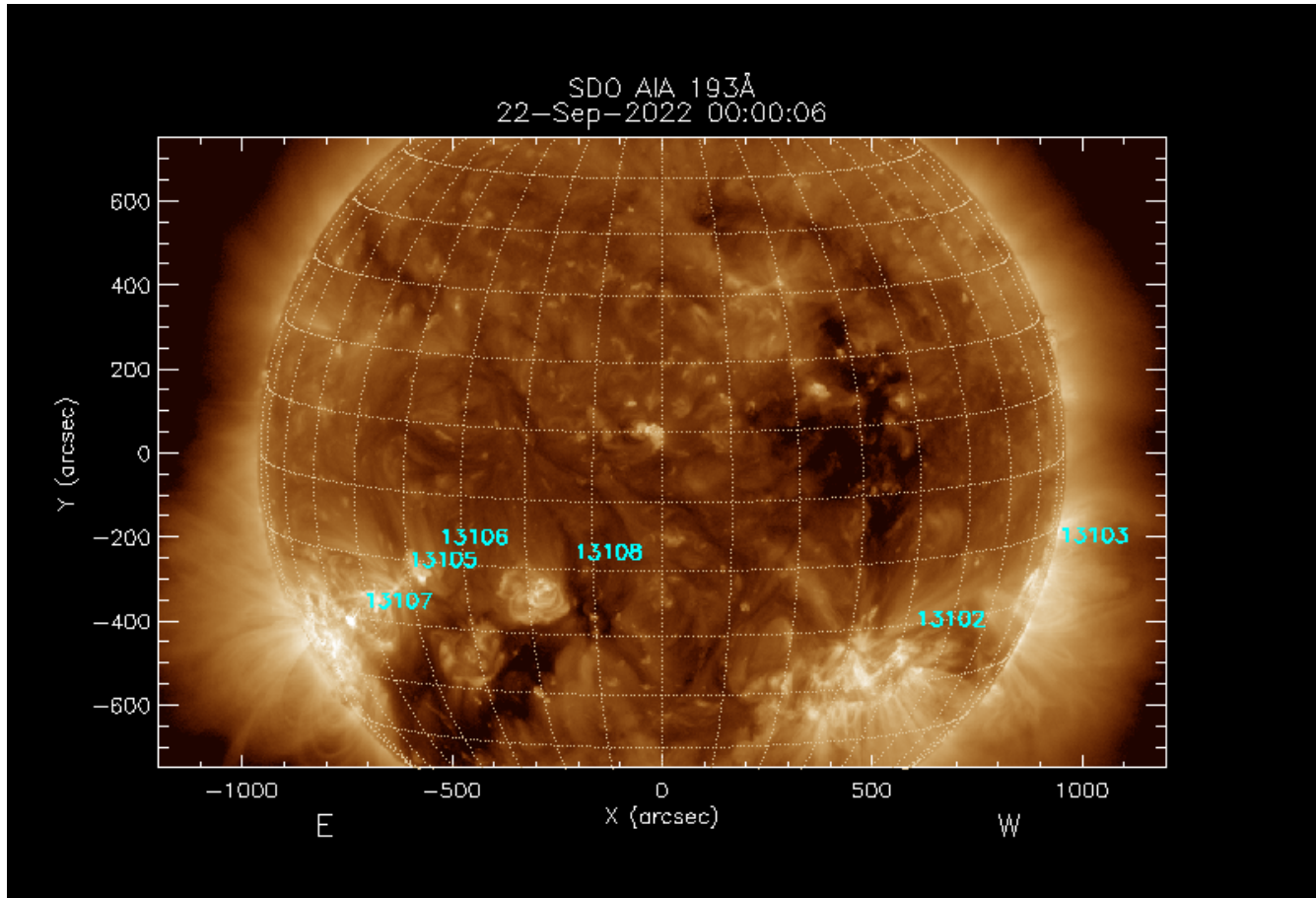
History of Active Region 13107

Description of the Active Region and a summary of flaring activity:

DATE	TYP	AREA	NSPOT	ZMCINT	MCLASS	C	M	X	TOTAL	SMON
2022-09-22	A	120	2	Dso	β	0	0	0	0	Link
2022-09-23	A	190	6	Cao	β	0	0	0	0	Link
2022-09-24	A	230	10	Cao	β	0	0	0	0	Link
2022-09-25	A	240	17	Fai	$\beta\gamma$	0	0	0	0	Link
2022-09-26	A	240	17	Eai	β	0	0	0	0	Link
2022-09-27	A	220	33	Esi	β	0	0	0	0	Link
2022-09-28	A	200	27	Eai	$\beta\gamma$	0	0	0	0	Link
2022-09-29	A	190	17	Eai	$\beta\gamma$	0	0	0	0	Link
2022-09-30	A	160	8	Eai	β	0	0	0	0	Link
2022-10-01	A	100	7	Cai	β	0	0	0	0	Link
2022-10-02	A	30	6	Cri	β	0	0	0	0	Link
2022-10-03	A	30	2	Hsx	α	0	0	0	0	Link

However, that listing does not include our flare. So, to be certain that this was the correct active region, we cross referenced the region number against this image (Figure 14), from helio.mssl.ac.uk, with the active regions labeled. This image was recorded about 17 hours after the flare occurred. The location of the region in this image is consistent with the location in the image above, factoring in the sun's rotation during that 17-hour span.

Figure 14: Flare region locations shortly after flare



Returning to Figure 11, the image recorded at the time of the flare, we checked the approximate coordinates indicated by the image against the GOES flare location file to confirm the precise location of the flare: (-847.3016, -422.4999).

The flare can be seen in this wavelength 304 movie we made at HelioViewer.org:

<https://helioviewer.org/?movieId=n1zn5>

Discussion

Lessons learned

We learned some important lessons in best practices for data science, as well as for machine learning and, specifically, time series modeling. First, we learned to put multiple sets of eyes on all code. The error in our date encoding likely would have been spotted and corrected earlier had the results been cross checked by another team member. We also learned to examine multiple ways of balancing our training set and multiple sizes of time windows. Another team which approached the problem similarly found greater success by optimizing the training set balance and window size.

Next steps

Were we to continue this research, we would conduct further research into best practices, including shapelet detection, for time series classification. We would also tune the balance between flares and non-flares in the dataset, as well as the time windows used. We would also continue to fine tune the parameters within the layers of the autoencoder model.

Conclusion

Our model successfully identified some of the flares recorded by the NASA algorithm, but failed to identify the majority of them. Still, our results do indicate that convolutional autoencoders hold promise for improving upon the current flare detection algorithm. Continued application of machine learning to solar flare detection should soon provide a more accurate method for detecting flares, providing future researchers with more complete data about solar flare activity.

Documentation

Github

<https://github.com/DMHwang/PDS>

Google drive

<https://drive.google.com/drive/folders/1iFOMoC0tp1ZC5TYKtZnpKtclSvCDUAFu?usp=sharing>

Websites

Data

GOES-R data: <https://www.ngdc.noaa.gov/stp/satellite/goes-r.html>

Flare Analysis

Record of flares on 9/21/22 from Solar Monitor:

https://solarmonitor.org/full_disk.php?date=20220922&type=hxrt_filter&indexnum=1

Record of the analyzed flare from the Lockheed Martin Solar and Astrophysics Laboratory:

https://www.lmsal.com/solarsoft/latest_events_archive/events_summary/2022/09/21/gev_20220921_0651/index.html

Image from the Mullard Space Science Laboratory showing the position of active region 13107, 17 hours after the analyzed flare:

http://helio.mssl.ucl.ac.uk/helio-vo/solar_activity/arstats/summary_dates/hio_FlareLocations_20220922.png

Summary of active region 13107:

http://helio.mssl.ucl.ac.uk/helio-vo/solar_activity/arstats/arstats_page5.php?region=13107

Appendix - David

We took a deep dive into the literature on solutions to time series problems for solar flares and other space- and weather-related phenomena. We found many instances of models for time series prediction, but little guidance for our problem of time series classification.

One method which seemed promising was detailed in the paper by Stefano Mauceri, *et al*: [Feature extraction by grammatical evolution for one-class time series classification](#), Genetic Programming and Evolvable Machines (2021) 22:267–295, DOI 10.1007/s10710-021-09403-x. The algorithm, available at https://github.com/spaghattix/FE_GE_TSC, uses computational methods to model anomalies in time series data and use them for binary classification - an exact match for our problem.

Using the algorithm on our data was a time-consuming and ultimately fruitless effort. Bugs in the code prevented successful execution. We reached out to the primary author, who generously gave of his time and, after several rounds of feedback, fixed the bugs. Unfortunately, the algorithm was computationally inefficient, running for hours on a small sliver of data. The poor accuracy of the results left little motivation to continue further with this approach.