Predicting Next Day Wildfire Spread

Rishabh Gupta, Shreyas Sai Raman, Zhen Wang

Abstract

Wildfire spread prediction is a crucial task in environmental modeling, with significant implications for public safety, emergency planning, and ecological conservation. In this project, we investigate the Next Day Wildfire Spread dataset—a large-scale, remote-sensing dataset curated for day-ahead prediction of wildfire expansion. Our objective is twofold: (1) to replicate the baseline models proposed in the original benchmark study, and (2) to conduct a comparative analysis of classical and deep learning approaches under varying hyperparameter settings. Specifically, we implement and evaluate five models—Logistic Regression, Random Forest, Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and a Convolutional Autoencoder—across multiple learning rates and batch sizes. Our findings provide insight into the sensitivity of these models to training configurations and underscore the potential of deep architectures, particularly convolutional and recurrent models, in capturing complex spatial-temporal wildfire dynamics.

1 Introduction

Wildfires are increasingly destructive due to climate change and human expansion, highlighting the need for accurate, short-term prediction models. To support this, Huot et al.[1] released the Next Day Wildfire Spread dataset—a large-scale public resource aggregating nearly a decade of remote-sensing data across the U.S., with fire masks and 11 environmental and human-related features at 1 km resolution. The dataset enables supervised learning for predicting wildfire spread from day t to t+1. In their benchmark, logistic regression, random forest, and a convolutional autoencoder were evaluated, with the autoencoder achieving the highest AUC-PR. Building on this, we implement five models—Logistic Regression, Random Forest, RNN, LSTM, and a CNN-based autoencoder—and mainly explore how learning rate and batch size affect performance. While full reproduction of the original results proved challenging, our analysis offers insights into both classical and deep learning approaches for wildfire prediction.

2 Technical Approach

We implemented five models to explore different paradigms for next-day wildfire spread prediction: Logistic Regression, Random Forest, Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), and a Convolutional Autoencoder (CNN-AE). Logistic Regression and Random Forest were chosen as classical baselines and implemented using Scikit-learn. For Random Forest, we followed the spatial neighborhood-based feature representation described in the original dataset paper, flattening 3×3 patches into 108-dimensional vectors. The deep learning models were built using PyTorch. The RNN and LSTM were designed to capture temporal patterns from sequential input data, with sequences passed through one or two recurrent layers followed by a fully connected prediction layer. To model spatial structure, we constructed a CNN-AE consisting of an encoder that compresses the input image and a decoder that reconstructs it. A final 1×1 convolution was applied for per-pixel binary classification of fire spread.

3 Experimental Results

3.1 Dataset

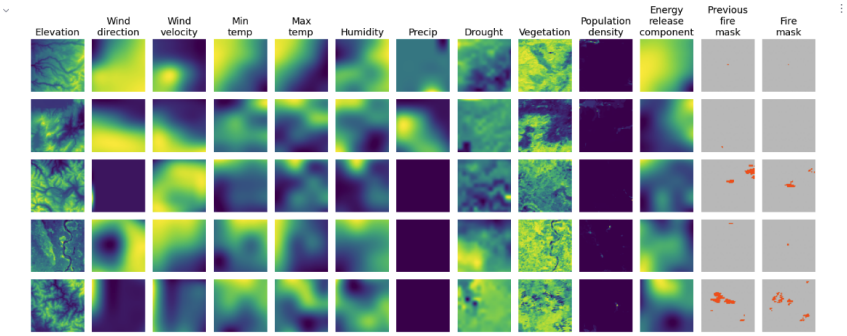


Figure 1: Dataset Visualization

We use the Next Day Wildfire Spread dataset [1], a large-scale public resource designed to support machine learning research on short-term wildfire prediction. Each data sample spans a 64×64 km^2 region in the continental U.S. and contains paired fire masks for two consecutive days (t and t+1), allowing for supervised learning of fire spread dynamics at a 1 km resolution. The dataset includes 11 environmental and anthropogenic features such as elevation, wind, temperature, humidity, vegetation index (NDVI), population density, and energy release component (ERC), aggregated from sources like MODIS [3], SRTM [4], GRIDMET [5], VIIRS [6], and GPWv4 [6].

To comply with project guidelines prohibiting the use of TensorFlow, we converted the official TFRecord files into a PyTorch-compatible format. This involved implementing a custom loader that preserved the spatial and temporal structure of the data while adapting it to PyTorch's tensor-based pipeline.

Although we initially attempted to reproduce the dataset generation pipeline using the authors' Google Earth Engine[2] (GEE) scripts, technical barriers—such as environment configuration and export constraints—prevented full replication. We therefore used the officially released TFRecords as the basis for our experiments.

3.2 Experiments

3.2.1 Logistic Regression

The logistic regression model demonstrates limited effectiveness on the validation set. While it achieves a moderate AUC-ROC of 0.6457, indicating some ability to distinguish between classes,

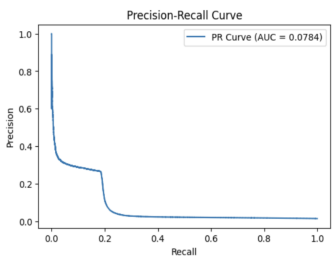


Figure 2: Precision vs Recall for Logistic Regression

the AUC-PR is very low at 0.0784 (Figure 2, suggesting poor performance in identifying positive cases, especially in the presence of class imbalance. Although the precision is relatively high at 0.5299, the recall is extremely low (0.0058), meaning the model identifies very few actual positives. Consequently, the F1 score is just 0.0114, reflecting an imbalanced trade-off between precision and recall. Overall, the model in its current form is too conservative and fails to generalize well for positive class detection, requiring threshold tuning, class imbalance handling, or a more expressive model to improve results.

3.2.2 Random Forest

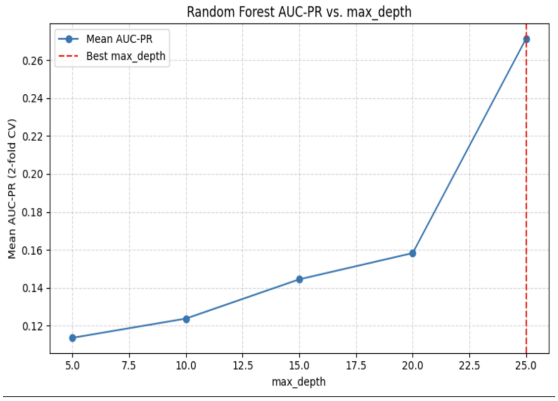
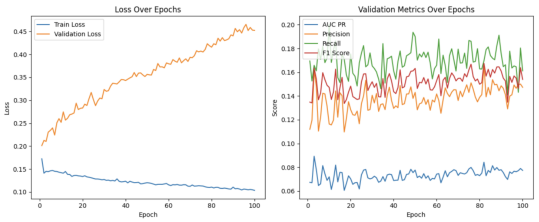


Figure 3: Mean AUC vs max-depth

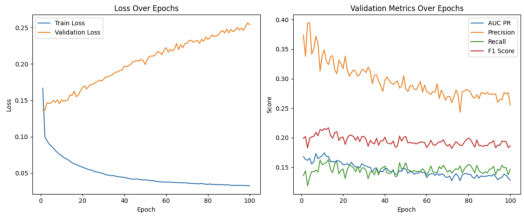
Tree Depth	Mean AUC PR (2-fold)
5	0.1101
10	0.1235
15	0.1435
20	0.1583
Unlimited	0.2713

Table 1: Impact of tree depth on Random Forest performance

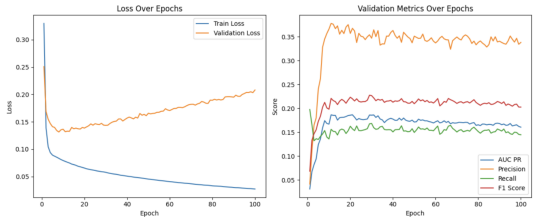
We performed a sweep over the max depth hyperparameter in the range [5, 10, 15, 20, None], using 2-fold cross-validation to select the best model based on validation AUC-PR. The best-performing configuration was found to be (max depth = None), meaning trees were allowed to grow without restriction. This setting yielded the highest mean AUC-PR on validation, suggesting that deeper trees helped capture the sparse and complex fire patterns in the data. The final model was retrained using the best parameters and evaluated on the test set, resulting in the following performance: AUC-PR(Test): 0.1530, Precision: 0.5107, Recall: 0.0060, F1 Score: 0.0119. While the model achieved high precision, the recall remained extremely low, indicating that the classifier was highly conservative — likely only predicting fire when very confident. This trade-off between precision and recall aligns with the findings in the original paper and highlights the difficulty of identifying rare fire pixels without sacrificing false positives.



(a) LR = 0.01



(b) LR = 0.001



(c) LR = 0.0001

Figure 4: Training curves and validation performance for RNN with varying learning rates

Learning Rate	AUC PR	Precision	Recall	F1-score
0.01	0.0774	0.1473	0.1612	0.1539
0.001	0.1280	0.2554	0.1466	0.1863
0.0001	0.1603	0.3379	0.1445	0.2024

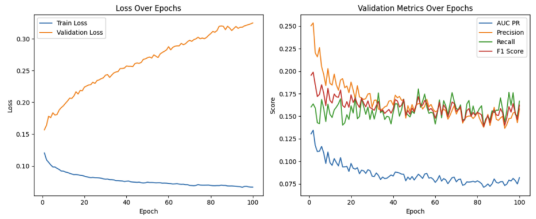
Table 2: Validation metrics across learning rates for RNN

3.2.3 Two Layer Recurrent Neural Network (RNN)

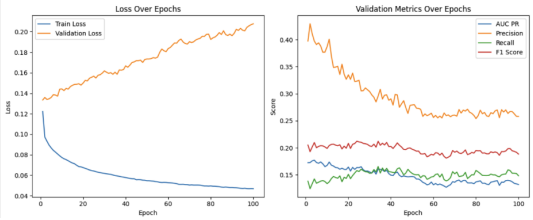
As shown in Figure 4a, with a learning rate of 0.01, the model converges rapidly but exhibits high validation loss volatility and validation metrics being very noisy. Significant overfitting and inadequate generalization are indicated by the validation loss's steady increase while the training loss drops off quickly.

When reducing the learning rate to 0.001 (Figure 4b), the validation loss still increases, but provides better control over overfitting than a learning rate of 0.01. Further reduction to 0.0001 (Figure 4c) yields the best overall performance, with loss showing minimal overfitting and good convergence.

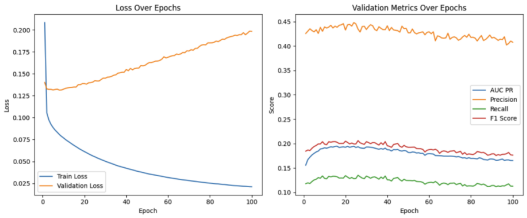
3.2.4 LSTM



(a) LR = 0.01



(b) LR = 0.001



(c) LR = 0.0001

Figure 5: Training curves and validation performance for LSTM with varying learning rates

Learning Rate	AUC PR	Precision	Recall	F1-score
0.01	0.0819	0.1580	0.1668	0.1623
0.001	0.1322	0.2580	0.1483	0.1884
0.0001	0.1649	0.4078	0.1124	0.1762

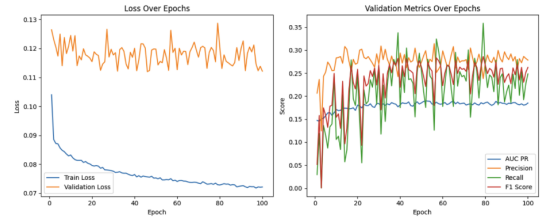
Table 3: Validation metrics across learning rates for LSTM

We train the LSTM model over 100 epochs with Adam optimizer on two T4 GPUs and evaluate performance using AUC-PR, F1-score, Recall and Precision. To improve performance and generalization, we conduct extensive hyperparameter tuning across learning rates (0.01, 0.001, 0.0001). Figure 5a illustrates that, similar to the RNN model, the model converges quickly at a learning rate of 0.01 but shows significant volatility in validation loss. Severe overfitting and poor generalization are indicated by the validation loss increasing slowly while the training loss rapidly declines.

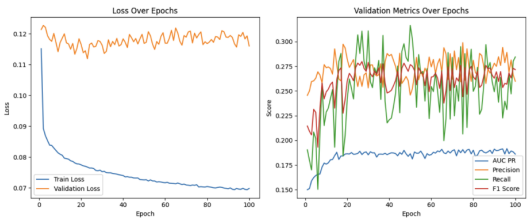
When reducing the learning rate to 0.001 (Figure 5b), like in RNN model, the validation loss still increases, but provides better control over overfitting than a learning rate of 0.01. Further reduction to 0.0001 (Figure 5c) yields the best overall performance, with training loss and validation loss decreasing more gradually and consistently.

From the results, we observe that a learning rate of 0.0001 provides the best overall generalization performance with the highest AUC PR and precision, and stable loss curves, even if the recall is slightly lower. It shows the most consistent and reliable behavior across all metrics, making it the best-performing model in this comparison.

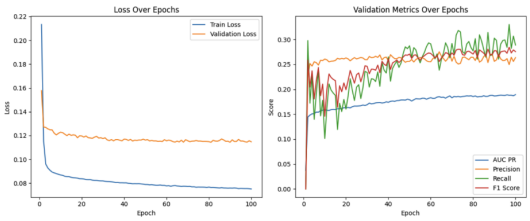
3.2.5 Convolutional Autoencoder



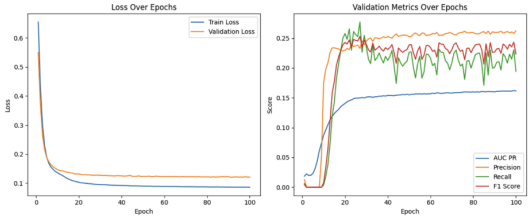
(a) LR = 0.01



(b) LR = 0.001



(c) LR = 0.0001

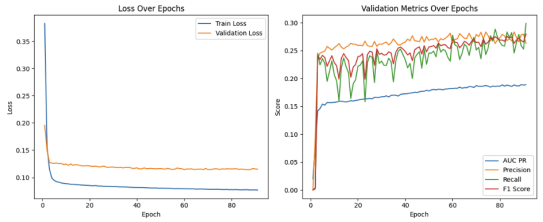


(d) LR = 0.00001

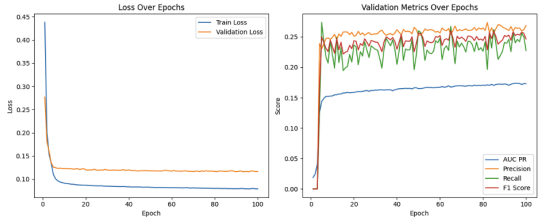
Figure 6: Training curves and validation performance for Convolutional Autoencoder with varying learning rates

Learning Rate	AUC PR	Precision	Recall	F1-score
0.01	0.1851	0.2784	0.2484	0.2626
0.001	0.1860	0.2598	0.2844	0.2715
0.0001	0.1892	0.2635	0.2887	0.2755
0.00001	0.1615	0.2622	0.1943	0.2232

Table 4: Validation metrics across learning rates for Convolutional Autoencoder



(a) Batch size = 64



(b) Batch size = 128

Figure 7: Training curves and validation performance for Convolutional Autoencoder with varying batch size

We train the model over 100 epochs with Adam optimizer on two T4 GPUs and evaluate performance using AUC-PR, F1-score, Recall and Precision. To improve performance and generalization, We conduct extensive hyperparameter tuning across learning rates (0.01, 0.001, 0.0001, 0.00001), batch sizes (32, 64,128), and the number of filters in the first convolutional block (16, 32).

Batch Size	AUC PR	Precision	Recall	F1-score
32	0.1892	0.2635	0.2887	0.2755
64	0.1887	0.2630	0.2988	0.2798
128	0.1729	0.2687	0.2275	0.2454

Table 5: Validation metrics across batch size for Convolutional Autoencoder

Filter Size	AUC PR	Precision	Recall	F1-score
16	0.1892	0.2635	0.2887	0.2755
32	0.1898	0.2807	0.2472	0.2629

Table 6: Validation metrics across first convolutional block filter sizes

Our experimental results demonstrate significant variations in model performance based on learning rate selection. As shown in Figure 6a, with a learning rate of 0.01, the model converges rapidly but exhibits high validation loss volatility. The training loss decreases steadily while validation metrics show considerable fluctuation, particularly in recall measurements.

When reducing the learning rate to 0.001 (Figure 6b), the results exhibit a similar pattern as in the previous case, only with slight improvement in validation metrics (Table 4). Further reduction to 0.0001 (Figure 6c) yields the best overall performance, with training loss steadily decreasing to approximately 0.075 and validation loss stabilizing around 0.118.

With the lowest learning rate of 0.00001 (Figure 6d), we observe a more smooth training and validation curves despite slower convergence in early epochs. However, it achieves lower performance across all metrics as shown in Table 4.

We also observe that a learning rate of 0.0001 provides the best overall balance among AUC-PR (0.1892), Precision (0.2635), Recall(0.2887), and F1-score(0.2755), indicating more consistent performance in detecting fire pixels. Also, across all experiments, the 0.0001 learning rate also demonstrates faster convergence speed in early epochs while maintaining stable performance throughout training.

We evaluated the impact of different batch sizes—32, 64, and 128—on the performance of our wildfire spread prediction model (Table 5). The model achieved the highest AUC-PR score of 0.1892 with a batch size of 32, indicating better overall ranking ability for identifying fire pixels. Interestingly, while batch size 64 had a slightly lower AUC-PR of 0.1887, it achieved the highest recall (0.2988) and F1-score (0.2798), suggesting a better balance between precision and recall for this setting. Conversely, increasing the batch size to 128 led to a noticeable drop in AUC-PR (0.1729) and F1-score (0.2454), despite a marginal gain in precision (0.2687). These results suggest that smaller to moderate batch sizes offer a better trade-off for this imbalanced classification task.

With fixed learning rate 0.0001 and batch size 32, we further investigate the architecture by exploring the effect of filter size in the first convolutional block. Before passing data through the encoder-decoder structure, we set an input convolutional layer to process the input data. This layer serves as an initial feature extraction stage.

As shown in Table 6, our experiments with filter sizes reveal interesting performance tradeoffs. For validation metrics, the larger filter size (32) demonstrates superior AUC-PR and precision, while the smaller filter size (16) yields better recall and F1-score. When evaluating on the held-out test set (Table 7), the configuration with 16 filters exhibits stronger performance in AUC-PR, recall, and F1-score. Only in precision does the 32-filter configuration maintain its advantage. This performance difference between validation and test sets suggests that while both configurations are effective, the 16-filter model offers better generalization capabilities across diverse wildfire scenarios.

Filter Size	AUC PR	Precision	Recall	F1-score
16	0.2516	0.3212	0.3780	0.3473
32	0.2451	0.3373	0.2472	0.2629

Table 7: Test metrics across first convolutional block filter sizes

Based on these comprehensive results, we selected the 16-filter configuration for our final model to optimize overall wildfire detection performance.

Model	AUC PR	Precision	Recall	F1-score
Logistic Regression	0.1306	0.4307	0.0052	0.0103
Random Forest	0.1530	0.5107	0.0060	0.0118
RNN	0.1786	0.3634	0.1671	0.2290
LSTM	0.1894	0.4433	0.1341	0.2059
CNN-AE	0.2516	0.3212	0.3780	0.3473

Table 8: Comparison of model performance on wildfire spread prediction

Finally, we tested the held-out test data with all trained models as shown in Table 8. Among classical models, Random Forest outperformed Logistic Regression in both AUC-PR and precision, though both suffered from extremely low recall. Deep learning models demonstrated significantly better balance across metrics. The RNN and LSTM models improved recall and F1-score, with LSTM achieving a higher precision (0.4433). The Convolutional Autoencoder (CNN-AE) achieved the highest overall performance, with an AUC-PR of 0.2516 and an F1-score of 0.3473, confirming its strength in capturing spatial patterns for fire prediction.

4 Conclusions and Future Work

In this study, we evaluated five machine learning approaches—Logistic Regression, Random Forest, RNN, LSTM, and a CNN Autoencoder—for next-day wildfire spread prediction. Due to resource limitations, we were unable to exactly replicate the experimental settings from the original benchmark. Nevertheless, our results confirm that the CNN Autoencoder achieves superior performance in capturing spatial features, aligning with prior findings. Hyperparameter tuning, particularly learning rate selection, played a crucial role in model performance, with 0.0001 consistently yielding the best results. Class imbalance remains a significant challenge, as reflected by the volatility in recall across all models. Future work could focus on completing the replication pipeline to enable more customized data generation—for example, adapting the dataset to new geographic regions, incorporating additional remote sensing inputs, or refining spatial and temporal resolutions. The modular structure of the original GEE codebase facilitates such extensions and opens pathways for broader applications in wildfire modeling.

References

[1] Fantine Huot, R. Lily Hu, Nita Goyal, Tharun Sankar, Matthias Ihme, and Yi-Fan Chen. Next Day Wildfire Spread: A Machine Learning Data Set to Predict Wildfire Spreading from Remote-Sensing Data. arXiv preprint arXiv:2112.02447, 2022.

[2] Gorelick, N., et al. (2017). Google Earth Engine: Planetary-scale geospatial analysis for everyone. Remote Sensing of Environment, 202, 18–27.

[3] Giglio, L., & Justice, C. O. (2015). MOD14A1 MODIS/Terra Thermal Anomalies/Fire Daily L3 Global 1km SIN Grid V006. NASA EOSDIS Land Processes DAAC.

[4] Farr, T. G., et al. (2007). The Shuttle Radar Topography Mission. Reviews of Geophysics, 45(2).

[5] Abatzoglou, J. T. (2013). Development of gridded surface meteorological data for ecological applications and modelling. International Journal of Climatology, 33(1), 121–131.

[6] Didan, K., & Barreto, A. (2018). VIIRS/NPP Vegetation Indices 16-Day L3 Global 500m SIN Grid V001. NASA EOSDIS Land Processes DAAC.

[7] Center for International Earth Science Information Network (CIESIN). (2016). Gridded Population of the World, Version 4 (GPWv4). NASA Socioeconomic Data and Applications Center (SEDAC).