GeoTorch: A Spatiotemporal Deep Learning Framework

Kanchan Chowdhury
Arizona State University
Tempe, Arizona, USA
kchowdh1@asu.edu

Mohamed Sarwat
Arizona State University
Tempe, Arizona, USA
msarwat@asu.edu

ABSTRACT

Deep learning frameworks, such as PyTorch and TensorFlow, support the implementation of various state-of-the-art machine learning models such as neural networks, hidden Markov models, and support vector machines. In recent years, many extensions of neural network models have been proposed in the literature targeting the applications of raster and spatiotemporal datasets. Implementing these models using existing deep learning frameworks requires nontrivial coding efforts from the developers because these extensions either are hybrid combinations of various categories of neural network models or differ extensively from state-of-the-art models supported by existing deep learning frameworks. Moreover, existing deep learning frameworks lack the support for scalable data preprocessing required to form trainable tensors from raw spatiotemporal datasets. To enable easy implementation of these neural network extensions, we present GeoTorch, a framework for deep learning and scalable data processing on raster and spatiotemporal datasets. Along with the state-of-the-art spatiotemporal models and ready-to-use benchmark datasets, we propose a data preprocessing module that allows the processing and transformation of spatiotemporal datasets in a cluster computing setting.

CCS CONCEPTS

• **Deep Learning Systems** → *PyTorch, PyTorch Geometric Temporal.*

KEYWORDS

spatiotemporal deep learning, satellite images, apache spark

ACM Reference Format:
Kanchan Chowdhury and Mohamed Sarwat. 2022. GeoTorch: A Spatiotemporal Deep Learning Framework. In *The 30th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '22), November 1–4, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3557915.3561036

1 INTRODUCTION

Deep learning frameworks such as PyTorch, TensorFlow, Keras, and MXNet ease the integration of deep learning by providing implementations of many state-of-the-art neural networks. PyTorch has recently gained increased popularity among these libraries because of its modular structure and easy Pythonic coding style. PyTorch ecosystem consists of various libraries which further enhance the functionality of PyTorch for specific domains. For example, PyTorch Geometric Temporal [8] provides algorithms for spatiotemporal signal processing. The abnormal growth of raster and spatiotemporal data recently resulted in directing the attention of Artificial Intelligence researchers toward applications such as traffic flow forecasting, bike and taxi volume prediction, land, buildings, trees and water classification, and raster image segmentation. Existing libraries in the PyTorch ecosystem and other deep learning frameworks suffer from the following limitations when they are applied to raster and spatiotemporal domains:

1) Spatiotemporal datasets can be divided into two main categories based on their tensor representation: grid-based and graph-based datasets. Existing libraries in the spatiotemporal domain such as PyTorch Geometric and Spektral can model only spatial dependency. Although some other systems such as PyTorch Geometric Temporal [8] and Dynamic GEM support both spatial and temporal dependency, these works cover only graph-based datasets and models, while grid-based models and datasets are left uncovered in all libraries. 2) Converting raw spatiotemporal datasets into trainable tensors requires significant preprocessing steps. Performing the preprocessing with Pandas and GeoPandas DataFrame results in slow preprocessing and memory error because of the large volume of these datasets. Usage of distributed spatial data processing systems such as Apache Sedona requires domain knowledge which limits machine learning practitioners to use only ready-to-use benchmark datasets. 3) CNNs are proved to be efficient for 2-channel grayscale and 3-channel RGB images, but raster images might consist of more than three spectral bands. There are some models efficient for modeling raster images, such as DeepSAT [1] and DeepSATV2 [6], which propose including handcrafted image features in the feature vector. Using existing deep learning frameworks to implement these models requires nontrivial manual efforts from the developers.

To alleviate these concerns, we propose GeoTorch, a library on top of PyTorch and Apache Sedona for data preprocessing and deep learning with raster and grid-based spatiotemporal datasets. Unlike PyTorch Geometric Temporal and Dynamic GEM, which cover graph-based datasets, our library focuses on grid-based spatiotemporal datasets along with raster datasets. GeoTorch contains benchmark datasets, state-of-the-art models, transformation operations, and data preprocessing functions for both raster and grid-based spatio-temporal domains. While datasets, models, and transforms modules run on PyTorch, the data preprocessing module runs on Apache Sedona, a cluster computing system for managing large-scale geospatial data on top of Apache Spark. Although the data preprocessing module runs on Apache Sedona, users of GeoTorch do not require an understanding of the coding syntaxes in Apache Sedona and PySpark. GeoTorch provides various Python functions to process and transform raster images and to convert raw spatiotemporal datasets into grid-based spatiotemporal tensors. Table

SIGSPATIAL '22, November 1–4, 2022, Seattle, WA, USA

ACM ISBN 978-1-4503-9529-8/22/11.
https://doi.org/10.1145/3557915.3561036

1 shows that our work is different from existing spatiotemporal deep learning frameworks in terms of supporting features.

Table 1: Features of Spatiotemporal Deep Learning Frameworks

Library	Spatial	Temporal	Grid	Raster	Scalable Preprocessing
PT Geometric	✓	✗	✗	✗	✗
Spektral	✓	✗	✗	✗	✗
Dynamic GEM	✓	✓	✗	✗	✗
PT Geometric Temporal	✓	✓	✗	✗	✗
Our Work: GeoTorch	✓	✓	✓	✓	✓

In summary, our contributions are as follows:

- We propose GeoTorch, a data preprocessing and deep learning library for raster and grid-based spatiotemporal datasets.
- Our deep learning module contains state-of-the-art ready-to-use benchmark datasets and models from the literature in both raster and spatiotemporal grid categories.
- Our preprocessing module allows users to generate grid-based tensors from raw spatiotemporal datasets and transform raster images in a distributed setting.
- We perform empirical evaluation on both the deep learning module and data preprocessing module and prove their effectiveness.

2 SYSTEM OVERVIEW

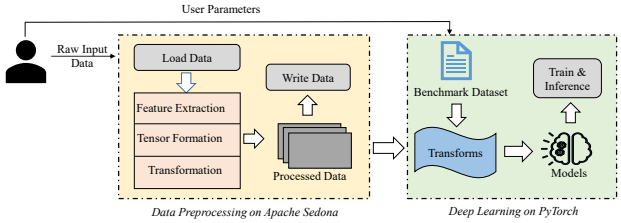


Figure 1: System Architecture of the Proposed Framework

Our proposed framework, GeoTorch, consists of two main modules as depicted in Figure 1 - Deep Learning and Data Preprocessing. While the deep learning module runs on PyTorch to make use of the GPU acceleration, the data preprocessing module utilizes the geospatial computation power of Apache Sedona and runs on PySpark to allow the processing in a cluster computing setting.

2.1 Deep Learning Module

Deep learning module offers three sub-modules for the raster and grid-based domains, including Datasets, Models, and Transforms.

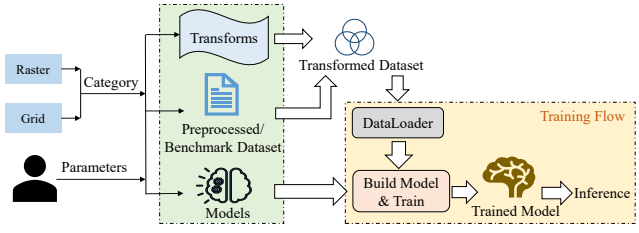


Figure 2: GeoTorch Deep Learning Module

2.1.1 GeoTorch Datasets. GeoTorch datasets module has two different packages for raster datasets and grid-based spatiotemporal datasets. Each of these packages contains easy-to-use benchmark datasets for widely used applications in the literature. GeoTorch datasets are created by extending *torch.utils.data.Dataset* class from PyTorch and uses the same iterator. That is why these datasets can be accessed and iterated similarly to PyTorch datasets and other datasets provided by libraries in the PyTorch ecosystem, such as Torchvision. Users can split a GeoTorch dataset into train and test sets and pass them to *torch.utils.data.DataLoader* to load samples as batches as well as in parallel by *torch.multiprocessing* workers. They can include transformation operations to the datasets using transforms offered by *geotorch.transforms* package or any user-defined transformations. Also, GeoTorch datasets can be loaded into either GPU and CPU based on client device configuration.

GeoTorch datasets module provides classes that allow defining any custom datasets instead of using only ready-to-use benchmark datasets. These classes are helpful in loading datasets that are transformed offline or created from raw spatiotemporal datasets using our preprocessing module described in Section 2.2. In the case of raster datasets, users can select the spectral bands they want to include in the feature vector. Besides, GeoTorch raster datasets provide the flexibility of including additional feature vectors extracted from raster images. For those users who lack domain knowledge on these spectral features, our raster datasets support automatically extracting some commonly used spectral features and including them in the feature vector. Our grid-based spatiotemporal datasets provide two flexible options for retrieving the samples during an iteration. 1) Data samples can be accessed in terms of closeness, period, and trend features as proposed in ST-ResNet [10]. 2) Samples can also be retrieved as sequences of history and prediction to enable the training of ConvLSTM and other sequence models.

2.1.2 GeoTorch Models. Similarly to the datasets module, models module also contains two packages - raster models and grid-based spatiotemporal models. These packages provide neural network layers for state-of-the-art raster and spatiotemporal models in the literature. Similarly to the neural network layers in PyTorch, GeoTorch models extend the torch.nn.Module class so that these models can be used as standalone neural network layers in Python as any other PyTorch neural network layers.

Each model is implemented efficiently using existing PyTorch neural network layers as building blocks. We minimize the number of public methods and mandatory parameters in all classes available as model layers and benchmark datasets. Few public methods and sufficient optional parameters allow users to customize the layers according to their requirements. Models can be run on either CPU or GPU. Both incremental and cumulative training approaches are supported. In the case of incremental training, model weights are updated after every batch, while cumulative training updates the weights once at the end of an epoch.

2.1.3 GeoTorch Transforms. GeoTorch transforms module provides transformation operations that can be applied to GeoTorch datasets during model training. Similar to Torchvision transformations, GeoTorch transforms can also be chained together using torchvision.transforms.Compose. Transformation operations can either be

passed as parameters when creating a dataset or be applied to samples when iterating over a dataset. GeoTorch raster transformation operations allow adding new feature vectors such as normalized difference index of two bands as a new band to the raster image. These features improve the efficiency of modeling raster images. GeoTorch also allows transformations offline before model training.

2.2 Data Preprocessing Module

Raw spatiotemporal datasets require a number of preprocessing steps in order to convert them into spatiotemporal tensors. Usually, these raw datasets are very large in size, and converting them into tensors requires a long processing time as well as high memory. Besides converting the geospatial coverage of the dataset into an $m \times n$ grid and desired temporal range into T time intervals, a user also needs to aggregate the data samples within each grid cell at every time interval. This aggregation process requires the application of time-consuming and memory-hungry spatial join queries. Preprocessing of these large raw datasets requires working on distributed and cluster computing geospatial data processing frameworks such as Apache Sedona. Besides spatiotemporal datasets, raster datasets also require preprocessing operations which can be classified into two categories - transformation operations and map algebra operations. Transformation operations are helpful in modifying the spectral bands of a raster image, such as normalizing a band, appending the normalized difference index between two bands as a separate band, deleting or inserting a band, etc. On the other hand, map algebra operations are used to extract features, such as getting different types of normalized difference indices, getting the mean, mode, modulas, square root of a band, etc. Since raster image datasets are also huge in volume, processing raster images on distributed systems can reduce the processing delay and memory-related errors. Performing the transformations offline in a distributed setting before model training instead of performing the same on-the-fly during model training can reduce the model training time and memory usage a lot.

GeoTorch preprocessing module contains two sub-modules: one for spatiotemporal grid data preprocessing and the other for raster image preprocessing. These sub-modules enable machine learning practitioners to process raster and spatiotemporal datasets in a distributed cluster computing setting without having domain and programming knowledge of geospatial cluster computing frameworks. Although GeoTorch preprocessing module runs on Apache Sedona, users can use its methods in a proper Pythonic way. In order to enable raster image transformation in a cluster computing setting, we contribute to Apache Sedona and add necessary satellite transformation and GeoTIFF image writing support to Apache Sedona. Spatial joins and other optimized operations implemented through Apache Sedona are a black box to the users.

3 EXPERIMENTAL EVALUATION

We evaluate our proposed framework, GeoTorch, on both raster and spatiotemporal models using some of our benchmark datasets. We also evaluate the model training time on CPU and GPU, besides verifying the effectiveness of the preprocessing module. The source code of GeoTorch is publicly available on GitHub [1].

[1]https://github.com/DataSystemsLab/GeoTorch

3.1 Experimental Settings

3.1.1 System Configuration. We perform all the model training experiments on a computer with an Intel NVIDIA GPU (GM107GL [Quadro K2200]) with 640 CUDA cores, 120 GB RAM, and a 4 TB hard drive. While most of the experiments are conducted enabling GPU, we run some of the experiments on CPU to show a comparison of runtime between CPU and GPU.

Table 2: Grid-Based Spatiotemporal Datasets

Dataset	Grid Shape	Time Interval	Time Duration
BikeNYC-DeepSTN[5]	21 × 12	1 Hour	01/04/2014 - 30/09/2014
TaxiNYC-STDN[9]	10 × 20	30 Minutes	01/01/2015 - 01/03/2015
BikeNYC-STDN[9]	10 × 20	30 Minutes	01/07/2016 - 29/08/2016
TaxiBJ21[4]	32 × 32	30 Minutes	November 2012, November 2014 and November 2015

3.1.2 Datasets. Widely used state-of-the-art datasets in raster and grid-based spatiotemporal categories have been made available as benchmark datasets in GeoTorch. Popular datasets used by grid-based spatiotemporal applications are listed in Table 2, while Table 3 lists the raster image datasets.

Table 3: Raster Image Datasets

Dataset	Type	Image Shape	Classes	Bands
SAT-6[1]	Multi-class Classification	28 × 28	6	4
SAT-4[1]	Multi-class Classification	28 × 28	4	4
EuroSAT[3]	Multi-class Classification	64 × 64	10	13
SlumDetection[2]	Binary Classification	32 × 32	2	4
38-Cloud[7]	Segmentation	384 × 384	-	4

3.2 Evaluating Deep Learning Module

We evaluate grid-based models on prediction task and raster models on satellite image classification and segmentation tasks.

3.2.1 Evaluating Spatiotemporal Prediction Task. Given the historical observations for a target attribute, such as bike flow, bike volume, traffic flow, traffic volume at various grid cells, the task is to predict the corresponding attribute values for a future timestep. We split each of these datasets into three parts: train, validation, and test sets. We use the records of the first 80% time intervals as the training dataset. Out of the remaining records, the earlier 10% records denote the validation dataset, while the test dataset contains the later 10% records. We initialize and train each model for five iterations, while each iteration consists of 100 epochs. We record the MAE and RMSE obtained by the models trained in each of 5 iterations and report the average of the results.

Prediction errors of various models on various datasets have been reported in Table 4. Reported results indicate that all models perform very close to each other on TaxiBJ21 [4] dataset. On the contrary, DeepSTN+ outperforms other models significantly on BikeNYC-DeepSTN dataset. The superior performance of DeepSTN+ and ST-ResNet over ConvLSTM model validates the usefulness of closeness, period, and trend features proposed by ST-ResNet [10] model. Also, performance of a model changes very slightly from one run to another run which proves the consistency of a model trained using GeoTorch.

Table 4: Evaluating Deep Learning Module and Preprocessing Module

Model	BikeNYC-DeepSTN		TaxiBJ21	
	MAE	RMSE	MAE	RMSE
ST-ResNet	2.912±0.090	7.317±0.227	0.044±0.004	0.073±0.005
DeepSTN+	2.325±0.056	6.085±0.133	0.019±0.003	0.032±0.006
ConvLSTM	4.655±0.116	11.832±0.254	0.082±0.006	0.120±0.009

(a) Predictive Performance of grid-based models

Model	Dataset	Application	Accuracy
DeepSAT V2	EuroSAT	Classification	94.070±0.208%
	SAT6	Classification	99.328±0.071%
SatCNN	EuroSAT	Classification	94.385±0.755%
	SAT6	Classification	98.921±0.100%
UNet	38-Cloud	Segmentation	97.341±0.217%
FCN	38-Cloud	Segmentation	96.907±0.331%

(b) Evaluating Accuracy of Raster Models

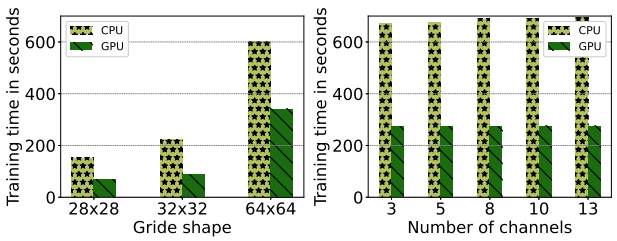
Count	Train with Transforms	Train with Pretransforms	Pretransforms
1	31302	22402	738
2	31896	22391	882
3	32188	22415	1020
4	32648	22396	1159
5	32977	22429	1276

(c) Training and Preprocessing Time in Seconds

3.2.2 Evaluating Models on Raster Image. We perform the satellite image classification task on two datasets: EuroSAT [3] and SAT6 [1], while the segmentation task is evaluated with 38-Cloud [7] dataset. For those datasets that do not have training and test sets predefined such as EuroSAT, We split each of them into 80% training set and 20% test set. The model training is performed in 5 iterations, each with 50 epochs, and the average results are reported in Table 4.

Reported results indicate that evaluated models perform very close to each other on both tasks, and none of the models can dominate on all datasets. Although DeepSAT V2 has fewer convolution layers than SatCNN, it shows equal competitive performance, which indicates the effectiveness of the feature fusion idea and handcrafted features proposed in DeepSAT V2.

3.2.3 Evaluating Runtime Performance. To evaluate the runtime, we measure the training time of a single epoch, varying the number of bands and grid size. We measure the training time by running the models separately on CPU and GPU. We train the SatCNN model for a single epoch with EuroSAT dataset. Figure 3 depicts the changes in the training time of an epoch depending on variations in the number of bands and grid size. From the observations, we can summarize that number of channels or bands does not affect the training time, while grid shape has a significant impact on the training time. Also, training time can be reduced drastically by running a model on GPU instead of CPU.



(a) Runtime vs Grid Shape **(b) Runtime vs No. of Channels**

Figure 3: Training Time for an Epoch on Various Grid Shapes and Number of Channels

3.3 Evaluating Preprocessing Module

We evaluate the effectiveness of our preprocessing module by performing the raster transformation operations before model training using the preprocessing module instead of performing preprocessing on-the-fly during training. We record the total preprocessing time for various counts of transformation operations and training time for model training with the transformed images. Separately, we also record the total training time while performing similar transformations during training. We repeat the process for 1, 2, 3, 4, and 5 transformation counts. Each transformation operation appends a normalized difference index to the image data. Table 4 reports the data preprocessing and model training time in various settings. It can be observed that the summation of pre-transformation time and model training time with pre-transformed data is much lower than the training time with transformation on the fly. In the case of transformation during training, training time increases with the increase in the number of transformations which proves the effectiveness of pre-transformation with our preprocessing module.

4 CONCLUSION

In this work, we presented our framework GeoTorch, a spatiotemporal deep learning framework for raster and grid datasets. Taking advantage of the existing PyTorch classes for neural networks, datasets, and transforms, GeoTorch provides classes for these modules in exactly the PyTorch way, allowing deep learning on satellite image datasets and grid-based spatiotemporal datasets which have not been covered by existing deep learning frameworks. Besides, preprocessing module of GeoTorch 1) allows scalable data preprocessing and transformation, which further helps in significant reduction in training time and 2) supports training and testing of models with raw spatiotemporal datasets instead of being limited to only ready-to-use benchmark datasets. We empirically evaluate the spatiotemporal prediction task, satellite image classification and segmentation tasks, along with showing how training time changes based on various input parameters on both CPU and GPU. Our empirical evaluation of preprocessing module proves the effectiveness of scalable data preprocessing before model training.

REFERENCES

[1] Saikat Basu, Sangram Ganguly, Supratik Mukhopadhyay, Robert DiBiano, Manohar Karki, and Ramakrishna Nemani. 2015. DeepSat: A Learning Framework for Satellite Imagery. In *SIGSPATIAL'15*.
[2] Federico Baylé. 2017. *Slum and Informal Settlements Detection.* https://www.kaggle.com/fedebayle/slums-argentina
[3] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian Borth. 2019. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* (2019).
[4] Weiwei Jiang. 2021. TaxiBJ21 : An open crowd flow dataset based on Beijing taxi GPS trajectories. *Internet Technology Letters* (2021).
[5] Ziqian Lin, Jie Feng, Ziyang Lu, Yong Li, and Depeng Jin. 2019. DeepSTN+: Context-Aware Spatial-Temporal Neural Network for Crowd Flow Prediction in Metropolis. *AAAI'19* 33 (2019), 1020–1027.
[6] Qun Liu, Saikat Basu, Sangram Ganguly, Supratik Mukhopadhyay, Robert DiBiano, Manohar Karki, and Ramakrishna R. Nemani. 2019. DeepSat V2: feature augmented convolutional neural nets for satellite image classification. *Remote Sensing Letters* 11 (2019), 156 – 165.
[7] Sorour Mohajerani and Parvaneh Saeedi. 2019. Cloud-Net: An End-To-End Cloud Detection Algorithm for Landsat 8 Imagery. 1029–1032.
[8] Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Alexander Riedel, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Guzmán López, Nicolas Collignon, and Rik Sarkar. 2021. PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models. In *CIKM '21*. 4564–4573.
[9] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Jessie Li. 2019. Revisiting Spatial-Temporal Similarity: A Deep Learning Framework for Traffic Prediction. In *AAAI'19*.
[10] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In *AAAI'17*. 1655–1661.