A Demonstration of GeoTorchAI: A Spatiotemporal Deep Learning Framework

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

Mohamed Sarwat
Wherobots Inc.
Scottsdale, Arizona, USA
mo@wherobots.com

ABSTRACT

This paper demonstrates GeoTorchAI, a spatiotemporal deep learning framework. In recent years, many neural network models have been proposed focusing on the applications of raster imagery and spatiotemporal non-imagery datasets. Implementing these models using existing deep learning frameworks, such as PyTorch and TensorFlow, requires nontrivial coding efforts from the developers because these models 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, a mandatory step for converting spatiotemporal datasets into trainable tensors. GeoTorchAI enables machine learning practitioners to implement spatiotemporal deep learning models with minimum coding efforts on top of PyTorch. It provides state-of-the-art neural network models, ready-to-use benchmark datasets, and transformation operations for raster imagery and spatiotemporal non-imagery datasets. Besides deep learning, GeoTorchAI contains a data preprocessing module that allows preparing trainable spatiotemporal vector datasets and the transformation of raster images 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. 2023. A Demonstration of GeoTorchAI: A Spatiotemporal Deep Learning Framework. In *Proceedings of the 2023 International Conference on Management of Data (SIGMOD '23).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Geospatial artificial intelligence applications have recently gained a lot of attention due to the abnormal rise of raster and spatiotemporal datasets. Examples include but are not limited to: traffic flow forecasting, bike/taxi volume prediction, land, buildings, trees, and water classification, as well as raster image segmentation. Many models have been proposed in the literature which are successful in spatiotemporal prediction and raster image classification tasks. These models represent extensions of commonly used neural network models, e.g., Convolutional Neural Network (CNN) and variants of Recurrent Neural Network (RNN), or hybrid models combining various types of neural networks.

Existing deep learning frameworks such as PyTorch, TensorFlow, Keras, MXNet and other libraries in their ecosystem suffer from the following limitations when they are used for implementing raster and spatio-temporal deep learning models:

- **Limitation 1:** Grid-based and graph-based datasets are the two basic divisions of spatio-temporal non-imagery datasets based on their tensor representation. Existing spatio-tempral deep learning libraries, such as PyTorch Geometric Temporal [5] and Dynamic GEM, support only graph-based datasets and models, leaving grid-based models and datasets unaddressed.
- **Limitation 2:** CNNs have been shown to be effective for two- and three-channel images, however raster images may have more than three spectral bands. Some effective raster image modeling algorithms, such DeepSAT [1] and DeepSATV2 [4], suggest integrating hand-crafted raster image features in the feature vector. Implementing these models using existing deep learning systems costs the developers nontrivial manual efforts.
- **Limitation 3:** Raw spatio-temporal datasets must undergo extensive preprocessing before they can be turned into trainable tensors. Due to the size of these datasets, preprocessing with Pandas and GeoPandas DataFrame results in slow preprocessing and memory errors. Because the usage of distributed geographic data processing systems like Apache Sedona necessitates domain expertise, machine learning practitioners rely only on benchmark datasets that are already ready for use.

This paper demonstrates GeoTorchAI[1] [2], a library on top of PyTorch and Apache Sedona that overcomes all the limitations discussed above. GeoTorchAI focuses on grid-based spatio-temporal datasets coupled with raster datasets, as opposed to PyTorch Geometric Temporal, Dynamic GEM, and TorchGeo, which cover either graph-based datasets or raster datasets. None of these existing systems support scalable data preprocessing, which has been integrated under GeoTorchAI. Table 1 shows how, in terms of supporting features, GeoTorchAI differs from other spatio-temporal deep learning frameworks. For both raster and grid-based spatiotemporal domains, GeoTorchAI includes benchmark datasets, state-of-the-art models, transformation operations, and data preprocessing functions. The data preprocessing module utilizes Apache Sedona, a cluster computing system for managing large-scale geospatial data on top of Apache Spark, while the deep learning module

[1] GeoTorchAI GitHub Repository: https://github.com/DataSystemsLab/GeoTorchAI

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '23, June 18–23, 2023, Seattle, WA
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

utilizes PyTorch. Users of GeoTorchAI do not need to be familiar with the PySpark and Apache Sedona coding syntaxes, even though the data preprocessing module runs on Apache Sedona. In order to filter and transform raster images and turn unprocessed spatio-temporal datasets into grid-based spatio-temporal tensors, GeoTorchAI offers a variety of Python functions. Since the preprocessing module runs on a cluster computing system, it is distributed and parallelized, similar to Apache Spark, and can reduce latency and memory errors. The deep learning module can be used in a fully PyTorch way targeting the applications like spatio-temporal traffic and weather prediction, satellite image classification, and satellite image segmentation.

Table 1: Features of Spatiotemporal Deep Learning Frameworks

Library	Spatial	Temporal	Grid	Raster	Scalable Preprocessing
PT Geometric	✓	✗	✗	✗	✗
Spektral	✓	✗	✗	✗	✗
TorchGeo	✓	✗	✗	✓	✗
Dynamic GEM	✓	✓	✗	✗	✗
PT Geometric Temporal	✓	✓	✗	✗	✗
GeoTorchAI	✓	✓	✓	✓	✓

We demonstrate four different features of GeoTorchAI in this paper. In the first demonstration, we prepare a spatiotemporal grid tensor from a large spatiotemporal raw dataset. Similarly to the first demo, the second demonstration also focuses on the preprocessing module by loading, transforming, and writing raster imagery datasets. The third and fourth demonstrations are based on the deep learning module of GeoTorchAI. The third demo scenario showcases the training process of a spatiotemporal deep learning model using GeoTorchAI. Finally, we apply a deep learning model trained on GeoTorchAI to classify satellite images.

2 SYSTEM OVERVIEW

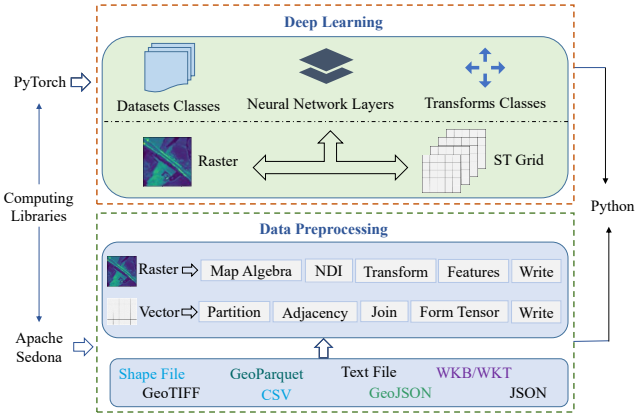

Figure 1: System Architecture of GeoTorchAI

GeoTorchAI consists of two main modules, as depicted in Figure 1 - Deep Learning and Data Preprocessing. The data preprocessing module uses Apache Sedona's geospatial computation power and enables processing in a cluster computing environment, while the deep learning module utilizes PyTorch's GPU acceleration.

2.1 Deep Learning Module

Datasets, Models, and Transforms are the three sub-modules available in the deep learning module of GeoTorchAI.

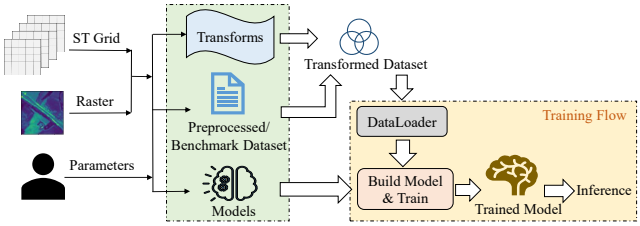

Figure 2: GeoTorchAI Deep Learning Module

2.1.1 GeoTorchAI Datasets. GeoTorchAI datasets module has separate packages for raster imagery datasets and grid-based non-imagery datasets. Each of these packages contains easy-to-use benchmark datasets for widely used applications in the literature. GeoTorchAI datasets extend *torch.utils.data.Dataset* class from PyTorch and use 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. Users can split a GeoTorchAI 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 *geotorchai.transforms* package or any user-defined transformations. Also, GeoTorchAI datasets can be loaded into either GPU and CPU based on client device configuration.

GeoTorchAI datasets module enables users to define any custom datasets instead of relying only on ready-to-use benchmark datasets. For this purpose, it provides classes, which are helpful in loading datasets that are transformed offline or created from raw spatiotemporal datasets using the GeoTorchAI preprocessing module described in Section 2.2. Users can choose which spectral bands they want to include in the feature vector for raster datasets. Additionally, GeoTorchAI raster datasets give users the freedom to include additional feature vectors extracted from raster images. GeoTorchAI raster datasets support automatically extracting some frequently used spectral features and including them in the feature vector for users who lack domain knowledge of these spectral features. All grid-based spatiotemporal datasets give users two adaptable choices for sampling during iterations. 1) As suggested by ST-ResNet [6], data samples can be accessed in terms of closeness, period, and trend features. 2) To aid in the training of ConvLSTM and other sequence models, samples can also be retrieved as histories and predictions.

2.1.2 GeoTorchAI Models. The models sub-module also contains separate packages for raster models and grid-based spatiotemporal models, just like the datasets module. These packages have neural network layers for state-of-the-art raster and spatiotemporal models that have been published. Similarly to the neural network layers in PyTorch, GeoTorchAI 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.

Models are implemented utilizing existing PyTorch neural network layers as building blocks. In all classes that are available as model layers and benchmark datasets, GeoTorchAI reduces the number of public methods and required parameters. Sufficient optional parameters are provided to allow users to customize the layers according to their requirements. Either the CPU or the GPU can be used to run models. GeoTorchAI supports both cumulative and incremental training. While cumulative training only updates the model weights once at the end of an epoch, incremental training updates the weights after each batch.

2.1.3 GeoTorchAI Transforms. GeoTorchAI transforms module offers transformation operations that can be used with raster datasets when training a model. GeoTorchAI transforms can be combined with any other transformation operations, just like the PyTorch transforms. Transformation operations can either be applied to samples when iterating over a dataset or be passed as parameters when creating a dataset. New features, such as the normalized difference index of two bands, can be added as a new band to the raster image using the transformation operations. These features make modeling raster images more effective. GeoTorchAI also supports offline transformations prior to model training.

2.2 Data Preprocessing Module

To create spatiotemporal tensors, raw spatiotemporal datasets need to go through a number of preprocessing steps. These raw datasets are typically very large in size, and creating tensors from them requires a lot of memory and processing time. In addition to converting the geospatial coverage of the dataset into an $m \times n$ grid and the desired temporal range into T time intervals, the user requires to aggregate the data samples within each grid cell at each time interval. Spatial join queries, which take a lot of time and memory, must be used for this aggregation process. These massive raw datasets must be preprocessed using distributed and cluster computing geospatial data processing frameworks like Apache Sedona. Raster datasets must also undergo preprocessing operations in addition to spatiotemporal datasets. These operations can be divided into transformation operations and map algebra operations. Transformation operations, such as normalizing a band, appending the normalized difference index between two bands as a separate band, deleting or inserting a band, etc., are useful for changing the spectral bands of a raster image. In contrast, map algebra operations are used to extract features, including obtaining various normalized difference indices, the mean, mode, modulas, and the square root of a band, etc. Raster image processing on distributed systems can decrease processing lag and memory-related errors because raster image datasets are also very large in volume. The amount of time and memory required for model training can be greatly decreased by performing the transformations offline in a distributed environment prior to training the model instead of performing the same on-the-fly during model training.

Two sub-modules, one for spatiotemporal grid data preprocessing and the other for raster image preprocessing, are available under GeoTorchAI preprocessing module. Machine learning practitioners who lack domain and programming knowledge of geospatial cluster computing frameworks can process raster and spatiotemporal datasets in a distributed cluster computing setting using these modules. Users can use the preprocessing methods in a proper Pythonic way, although the preprocessing module runs on Apache Sedona. GeoTorchAI adds necessary satellite transformation functions and GeoTIFF image writing support to Apache Sedona to facilitate scalable raster image transformation. Spatial joins and other optimized operations implemented through Apache Sedona are a black box to the users. Figure 3 depicts a few examples of raster image preprocessing.

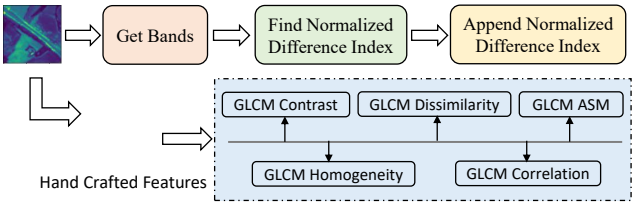

Figure 3: Raster Image Preprocessing Example

3 DEMONSTRATION SCENARIOS

We demonstrate GeoTorchAI using four different use cases which are described in this section. For each use case, GeoTorchAI provides an interface to load the data, set up the parameters, and display the outcome. A few screenshots of the interface are provided in Figures 4 and 5. The interface allows the users to perform data preprocessing, training, and testing interactively. An initial prototype of the demonstration[2] using the interface has been made available.

3.1 Use Case 1: Preparing Trainable Tensor

Consider an application that uses the New York City taxi trip dataset[3] to train a model which predicts the number of taxi pickups at various locations in New York City for future timesteps. For this purpose, the raw dataset of more than 1 billion taxi trip records needs to be converted into a trainable tensor. In this use case, we discuss the steps for forming a spatiotemporal tensor from the raw NYC taxi trip dataset. The tensor represents the spatial coverage of NYC City as an $h \times w$ grid where h and w stand for the height and width of the grid. The final tensor will contain the number of taxi pickups in each cell of the grid at various time intervals. The first step is to form the $h \times w$ grid by loading the taxi zone shape file via *load_geo_data* and calling the method *generate_grid_cells* under the class *geotorchai.preprocessing.grid.SpacePartition*. The next step is to load the *CSV* file, which contains the taxi trip information. Most of the remaining steps can be completed with various methods available in *STManager* class under *geotorchai.preprocessing.grid* package. Methods such as *trim_on_datetime*, *get_unix_timestamp*, and *add_temporal_steps* can be used to generate required time intervals from the pickup times. Latitudes and longitudes of pickup locations need to be converted to geometrical point objects with the help of the *add_spatial_points* method. The next step is to count the number of pickup locations in various cells at each time interval using the method *aggregate_st_dfs*. The last step is to convert the aggregated DataFrame into an array, with a shape similar to the

[2] https://kanchanchy.github.io/files/video/Demo-GeoTorchAI.mp4
[3] https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

target spatiotemporal tensor, utilizing the method *get_st_array*. Machine learning practitioners can load the returned spatiotemporal array into GeoTorchAI as a tensor with the *Processed* Dataset class, which is available under the package *geotorchai.datasets.grid*.

3.2 Use Case 2: Raster Data Preprocessing

The second use case of the demonstration focuses on transforming raster data using GeoTorchAI. An end-to-end raster processing example can be outlined as a series of the following simple steps. 1) Loading raster images by calling *load_geotiff_image* method, 2) Calling the required methods from *RasterProcessing* depending on the target transformation. A few examples include *get_normalized_band* to normalize the values of a particular band, *append_normalized_difference_index* to append the normalized difference index of two separate bands as a new spectral band, *mask_band_on_greater_than* to mask the values of a spectral band based on an upper threshold, etc. 3) Writing the transformed images by calling *write_geotiff_image* method.

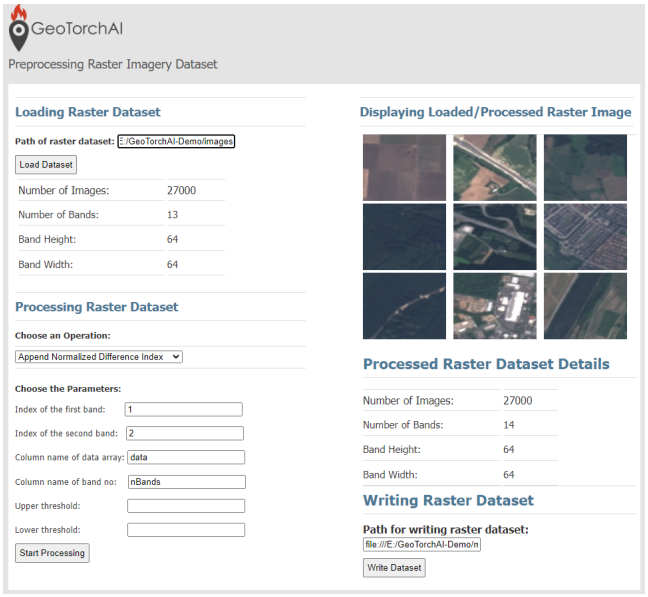

Figure 4: Raster Data Preprocessing Interface

3.3 Use Case 3: Raster Model Training

An end-to-end model training pipeline includes loading the dataset, defining the neural network model and training parameters, followed by model training. Let us discuss the model training pipeline on GeoTorchAI for a spatiotemporal application - classifying EuroSAT images [3] using DeepSAT V2 [4] model. The first step is to load the EuroSAT dataset used by instantiating *EuroSAT* class from GeoTorchAI package *geotorchai.datasets.raster*. If the dataset needs to be downloaded into a local directory, it can be done by setting the parameter *download* to *True*. Raster image-based deep learning models such as DeepSAT V2 propose several handcrafted image features to boost the accuracy of the trained model. Those features can be included automatically in the model training by setting the *include_additional_features* parameter to *True*. Later, we initialize DeepSAT V2 as our desired raster model from the package *geotorchai.models.raster*. Model initialization is followed by setting model hyperparameters which include loss function, optimizer, learning rate, and the number of epochs. During each epoch iteration, model training happens with a forward pass followed by a backward pass. In the forward pass, we pass the data samples of the training dataset to the model and get the output. The backward pass updates the model weights based on the loss between the model output and the ground-truth output.

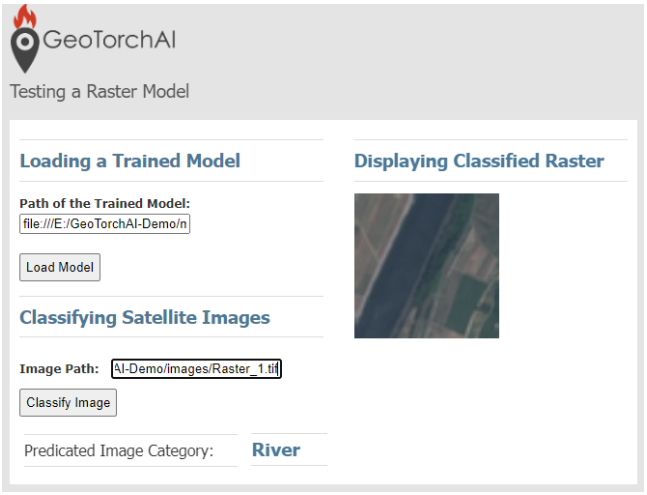

Figure 5: Raster Model Testing Interface

3.4 Use Case 4: Raster Image Classification

The fourth use case of the demonstration classifies raster images using a model trained with GeoTorchAI. The first step is to load a model that has been trained to classify raster images. We load the DeepSAT V2 model trained as part of use case 3 in Section 3.3 for our demonstration purpose. It should be noted that other models trained for raster image classification can also be loaded instead of DeepSAT V2. After loading the model, the next step is to select a raster image for classification. The model will predict the class of the selected image. The shape of the image should be similar to that of images, which have been used to train the model. Classifications can also be done in batches instead of classifying a single image.

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] Kanchan Chowdhury and Mohamed Sarwat. 2022. GeoTorch: A Spatiotemporal Deep Learning Framework *(SIGSPATIAL '22)*. Article 100.
[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] 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.
[5] 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.
[6] Junbo Zhang, Yu Zheng, and Dekang Qi. 2017. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. In *AAAI'17*. 1655–1661.