Deep Learning with Spatiotemporal Data: A Deep Dive into GeotorchAI

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

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

***Abstract*—In recent years, numerous neural network models have been put forth, with an emphasis 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 although these deep learning frameworks support the implementation of various state-of-the-art machine learning models, such as neural networks, hidden Markov models, and support vector machines. This is due to the fact that the models emphasized on spatiotemporal applications 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. Considering the limitations of existing deep learning frameworks, we present GeoTorchAI, a framework for deep learning and scalable data processing on raster imagery and spatiotemporal non-imagery datasets. 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 and a *DFtoTorch Converter* module that enable the formation of trainable spatiotemporal vector datasets and the mapping of preprocessed DataFrames into PyTorch tensors, respectively.**

***Index Terms*—spatiotemporal deep learning, raster images**

I. Introduction

Due to the rapid usage of GPS-enabled devices by moving objects and the availability of traffic sensors and satellite devices, the volume of raster and spatiotemporal datasets is growing at an unprecedented rate. 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, weather forecasting, bike/taxi volume prediction [1], traffic speed prediction, crowd flow prediction [2], land, buildings, trees, and water classification [3], slum and urban image classification, as well as raster image segmentation [4]. 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.

Deep learning frameworks such as PyTorch [5], TensorFlow [6], Keras [7], and MXNet [8] ease the integration of deep learning by providing implementations of many state-of-the-art neural networks. PyTorch [5] 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, Torchvision [9] extends the functionality of PyTorch for manipulating images. However, 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 spatiotemporal models discussed in the previous paragraph:

- **Limitation 1:** Spatiotemporal datasets can be divided into two main categories based on their tensor representation: grid-based and graph-based datasets [10]. Existing spatiotemporal deep learning libraries, such as PyTorch Geometric Temporal [11] and Dynamic GEM [12], 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-channel grayscale and three-channel RGB images. However, raster images may have more than three spectral bands. Some effective raster image modeling algorithms, such as DeepSAT [13] and DeepSATV2 [14], 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:** Converting raw spatiotemporal datasets into trainable tensors requires extensive preprocessing steps, including time-consuming and memory-hungry spatial join and data aggregation operations. Because the usage of distributed spatial data processing systems like Apache Sedona [15] necessitates domain expertise, machine learning practitioners rely only on ready-to-use benchmark datasets. To the best of our knowledge, there are no existing deep learning systems that allow users to create trainable tensors from spatiotemporal datasets in a scalable and Pythonic way without demanding domain expertise.
- **Limitation 4:** The extraction of satellite image features on the fly during the training slows down the training step. On

the contrary, extracting these features in a distributed and parallel setting before model training can boost the training speed considerably. Extracting the features offline before the training helps in reusing the features of a dataset every time these features are used to train a model.

- **Limitation 5:** The results of scalable preprocessing are distributed DataFrames that are unsuitable for direct use in PyTorch, requiring conversion into tensors. This conversion process often requires time-consuming data collection into the master node, which may also result in memory errors on the master node.

In this work, we discuss the design and development details of GeoTorchAI, which alleviates all the limitations discussed above. GeoTorchAI is a library on top of PyTorch and Apache Sedona for data preprocessing and deep learning with raster and grid-based spatiotemporal datasets. GeoTorchAI focuses on grid-based spatiotemporal datasets coupled with raster datasets, as opposed to PyTorch Geometric Temporal [11], Dynamic GEM [12], and TorchGeo [16], which cover either graph-based datasets or raster datasets. None of these existing systems support scalable data preprocessing, which has been integrated under GeoTorchAI. GeoTorchAI contains benchmark datasets, state-of-the-art models, transformation operations, and data preprocessing functions for both raster and grid-based spatiotemporal domains. While datasets, models, and transforms modules run on PyTorch, the data preprocessing module runs on Apache Sedona [15], 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 GeoTorchAI do not require an understanding of the coding syntaxes in Apache Sedona and PySpark. GeoTorchAI provides various Python functions to process and transform raster images and to convert raw spatiotemporal datasets into grid-based spatiotemporal tensors. 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 applications like spatiotemporal traffic and weather prediction, satellite image classification, and satellite image segmentation. To address the fifth limitation, we propose an intermediate *DFtoTorch Converter* module in GeoTorchAI, which takes the preprocessed DataFrame and can return the rows as batches in terms of PyTorch tensors during model training and inference. We empirically evaluate GeoTorchAI models for three applications: spatiotemporal prediction, satellite image classification, and satellite image segmentation. In addition, we evaluate the training time of models on both GPU and CPU. In the end, we conduct experiments to verify the efficiency of the GeoTorchAI data preprocessing module.

This work extends our initial study [17], [18], featuring significant advancements not explored in prior publications. These enhancements include: 1) supporting lots of new datasets and models in the deep learning module, 2) introducing the new *DFtoTorch Converter* module to map preprocessed DataFrame into PyTorch tensors, 3) releasing a new grid-based spatiotemporal traffic prediction dataset, 4) conducting new experiments on the preprocessing module to evaluate the effectiveness of end-to-end preprocessing and deep learning, and 4) evaluating deep learning module on a new application featuring new models and datasets. In summary, our contributions are as follows:

- We propose GeoTorchAI, a deep learning and scalable data processing library for raster and spatiotemporal datasets.
- We propose a deep learning module which contains state-of-the-art ready-to-use benchmark datasets, transforms, and models in both raster and spatiotemporal grid categories.
- We release a new grid-based spatiotemporal traffic prediction dataset, YellowTrip-NYC, based on the number of taxi pickups and dropoffs. The dataset is available on the GeoTorchAI GitHub repository.
- We perform an empirical evaluation on both the deep learning module and data preprocessing module of GeoTorchAI. Our evaluation shows that performing data transformation before model training with our preprocessing module can speed up the training process considerably.
- We perform an end-to-end evaluation of GeoTorchAI by forming a spatiotemporal tensor dataset with the GeoTorchAI preprocessing module and using the same dataset to train models with the GeoTorchAI deep learning module.

II. Background and Related Works

Before proceeding with the details of the GeoTorchAI framework, we define a few important concepts along with discussing the current state of deep learning with raster and spatiotemporal datasets.

A. Spatiotemporal Tensor Representation

Tensors are generalizations of vectors and matrices to potentially higher dimensions [19]. For a multivariate dataset, assume that there are a total of M variables in the dataset. The complete data is recorded for P locations over T timesteps. This dataset can be represented as a tensor $\mathcal{X} \in R^{TxPxM}$. Each location $P_i \epsilon P$ can either be a grid cell, polygon object, or graph node, depending on the type of the dataset. Spatiotemporal tensors can represent both spatial and temporal dependencies which we will discuss next.

1) Representing Temporal Dependencies: Temporal dependency in a spatiotemporal tensor is represented in one of two approaches: sequentially and periodically. Sequential representation is used mostly where the total time interval between the initial and end timestamps is equally divided into several time slots. Since changes in time-series data happen sequentially as time passes, adjacent timestamps are indexed in a consecutive manner so that the representation can hold the temporal correlation. In the case of periodical representation, the temporal dimension is indexed in a periodic manner, such as daily, weekly, and monthly. This type of representation is effective when attribute values at a time index show a similar pattern periodically. For instance, in the case of a traffic flow

dataset, the number of traffic might be lower than usual during weekends.

2) Representing Spatial Dependencies: The process of capturing spatial dependency in a spatiotemporal tensor depends on the type of the dataset. Spatial dependency is captured by an adjacency matrix in the case of a graph-based dataset, while grid-based datasets capture spatial dependencies through an image-like representation. Since GeoTorchAI focuses on grid-based spatiotemporal datasets, we detail the spatial dependency of grid-based tensors only. In the case of grid-based representation, the whole area covered by the dataset is considered a two-dimensional unit. The full spatial unit is converted into a grid-like structure by partitioning both the x-axis and y-axis into equal-sized slots such that all cells in the grid become equal, although slot sizes along the x-axis and y-axis might be different. Spatial partitions that are located adjacent to each other in the space also maintain their adjacency in the grid. This is also called an image-like representation, where the attribute value in each cell can be considered a pixel value. For multivariate datasets, each attribute is considered a channel in an image. Since CNNs are efficient in capturing spatial dependencies of images, this image-like grid representation is used to capture spatial dependencies with CNNs. For grid-based datasets, the representation of spatiotemporal tensor $\mathcal{X}^{grid}$ is changed to $\mathcal{X}^{grid} \in R^{TxHxWxC}$, where T, H, W, and C stand for the number of timesteps, grid height, grid width, and the number of channels/features, respectively.

B. Deep Learning with Grid-Based Spatiotemporal Datasets

The efficiency of a deep learning model in modeling spatiotemporal prediction tasks depends on its capability to capture spatial and temporal dependencies in the dataset. Most of the models [20], [1], [21], [22], [23], [24] capture the temporal dependency by the use of the variants of RNN - Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU). Some models [1], [21], [22], [23], [25], [26] introduce CNN's enhanced version TCN and attention layers in parallel with RNN variants to model temporal dependency and demonstrate superior performance [10]. As discussed in Section II-A2, CNN is used to capture spatial dependency by most of the models [2], [20], [1], [27], [21], while some other models capture the spatial dependency with attention layers [22], [24], [26].

Three types of representations of training datasets used by grid-based spatiotemporal tensors are discussed below:

Basic Representation: The most basic approach of a grid-based spatiotemporal model is to use the grid at timestep i as the data and the grid at timestep $i+t$ as the label. The model is trained to make predictions for timestep $i+t$ by looking at the data at timestep i. Here, t is known as the lead time. CNNs can be used to train this type of model. The prediction accuracy of these models is limited because predictions at a timestep are made by looking at the history of only one previous timestep, while the ground truth might depend on multiple timesteps.

Sequential Representation: To remedy the drawbacks of this basic representation, a hybrid model, ConvLSTM [28], is proposed that models the spatial and temporal dependencies using CNNs and LSTMs, respectively. The training procedure of this model takes two lengths as inputs - history length and prediction length. Denoting history length and prediction length as t_1 and t_2 respectively, this model uses data at t_1 timesteps to predict the labels at the next t_2 timesteps. Figure 1 depicts this representation for history length $n-1$ and prediction length 1. This representation outperforms the basic presentation because features at a particular timestep depend the most on its immediate previous timesteps. However, this representation is limited by the fact that attribute values at a timestep depend not only on its immediate previous timesteps but also on the timesteps of the previous day during the same period as well as external factors such as weekends and weather conditions.

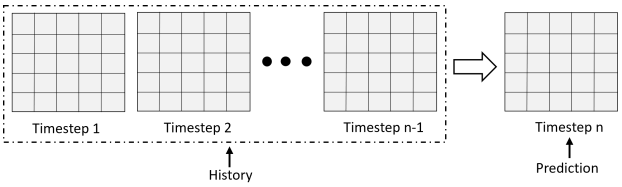


Fig. 1: Sequential Representation of Spatiotemporal Datasets

Periodical Representation: To further improve the prediction accuracy, an advanced representation is proposed by ST-ResNet [2], which is further inherited and improved by other models. ST-ResNet concatenates the channels in each timestep by converting (T, H, W, C) shaped tensors into (H, W, T*C) shaped ones for the purpose of capturing spatial dependencies with CNNs. It models the temporal dependency with three features named $Closeness$ (most recent observations), $Period$ (daily periodicity), and $Trend$ (weekly trend) [10]. Long-range spatial dependency between grid cells is modeled by constructing deep CNN networks with residual learning. Since ST-ResNet uses CNNs to capture both spatial and temporal dependency, it has to convert video-like tensors to image-like tensors, which is a limitation. Other models, such as STDN and DMVST-Net, address this issue by employing LSTM to connect with a CNN at each timestep [10].

C. Deep Learning with Raster Datasets

Deep learning with raster datasets usually aims at tasks such as classification and segmentation of raster images and change detection in raster images. Similarly to RGB or grayscale images, raster image-based tasks are also modeled with CNNs because raster images can also be represented similarly to RGB and grayscale images with the tensor representation $\mathcal{X} \in R^{HxWxC}$ where H, W, and C stand for image height, image width, and the number of channels/bands, respectively. The difference is that the number of spectral bands, C, can be more than 3 for raster images, while it is 2 and 3 for grayscale and RGB images, respectively. Figure 2 shows the thirteen spectral bands of a sample satellite image representing a forest from the EuroSAT dataset [3].

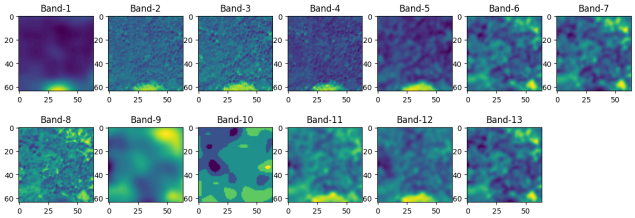


Fig. 2: Spectral Bands of an Image from EuroSAT Dataset

The efficiency of raster image modeling tasks depends on the selection of appropriate spectral bands. Deep learning models such as Fully Convolutional Networks [29] and UNet [30] have shown efficiency in modeling satellite image segmentation tasks. For the classification task, SatCNN [31] proposes a deep convolutional neural network that shows state-of-the-art performance. To enhance the accuracy of raster image classification tasks, DeepSat [13] proposes including some normalized features extracted from raster images in the feature vector. Basu et al. [32] prove that CNNs cannot learn Haralick feature representations by themselves. Utilizing this theory, Deepsat V2 [14] extends DeepSat by enhancing the architecture of a CNN to include handcrafted features and outperforms the classification accuracy of earlier works.

D. Spatiotemporal Deep Learning Frameworks

Deep learning frameworks such as PyTorch, TensorFlow, Keras, and MXNet are generic to all domains and are not fine-tuned for spatiotemporal datasets. There are several libraries in the ecosystem of these frameworks which contain advanced support for spatiotemporal datasets. The features supported by these popular libraries are listed in Table I. Among these libraries, Geometric2DR [33], PyTorch Geometric [34], TF Geometric [35], GEM [36], and StellarGraph [37] can only model spatial dependency and do not support temporal dependency. Two other libraries, Dynamic GEM [12] and PyTorch Geometric Temporal [11], can model temporal dependency, but these libraries only support graph-based spatiotemporal datasets and cannot be applied to raster and grid-based domains. Another framework, TorchGeo [16], supports only raster imagery datasets. None of these libraries support scalable processing of raster and spatiotemporal datasets. So, our work is completely different from these works because we focus on supporting raster and grid-based domains. Also, our work supports scalable and distributed preprocessing of spatiotemporal datasets, a missing feature in all other libraries.

III. System Overview

Our proposed framework, GeoTorchAI, consists of three main modules - *Deep Learning*, *Data Preprocessing*, and *DFtoTorch Converter*. Functions and classes offered by each of these modules can be used in Python. While the deep learning module runs on PyTorch [5] to make use of the GPU acceleration, the data preprocessing module utilizes the geospatial computation power of Apache Sedona [15] and runs on PySpark to allow the processing in a distributed cluster computing setting. In the case of scalable deep learning with the dataset preprocessed in a distributed manner, *DFtoTorch Converter* module works as an intermediate module between the two in order to convert the preprocessed DataFrame into trainable PyTorch dataset. Figure 3 depicts the system architecture of GeoTorchAI.

TABLE I: Features Supported by Spatiotemporal Deep Learning Frameworks

Library	Spatial	Temporal	Grid	Raster	Scalable Preprocessing
Geometric2DR [33]	✓	✗	✗	✗	✗
PT Geometric [34]	✓	✗	✗	✗	✗
TF Geometric [35]	✓	✗	✗	✗	✗
GEM [36]	✓	✗	✗	✗	✗
Spektral [38]	✓	✗	✗	✗	✗
TorchGeo [16]	✓	✗	✗	✓	✗
Dynamic GEM [12]	✓	✓	✗	✗	✗
PT Geometric Temporal [11]	✓	✓	✗	✗	✗
Our Work: GeoTorchAI	✓	✓	✓	✓	✓

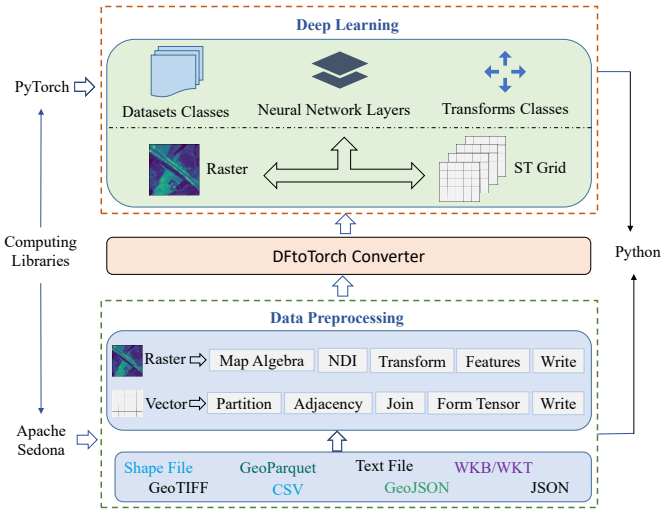


Fig. 3: System Architecture of the Proposed Framework

A. Deep Learning Module

The deep learning module offers three sub-modules for the raster and grid-based domains, including Datasets, Models, and Transforms, as depicted in Figure 4. This section provides an overview of these sub-modules.

1) GeoTorchAI Datasets: GeoTorchAI 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. GeoTorchAI 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 [9]. 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 the $geotorchai.transforms$ package or any user-defined transformations. Also, GeoTorchAI datasets can be loaded into either GPU or CPU based on client device configuration.

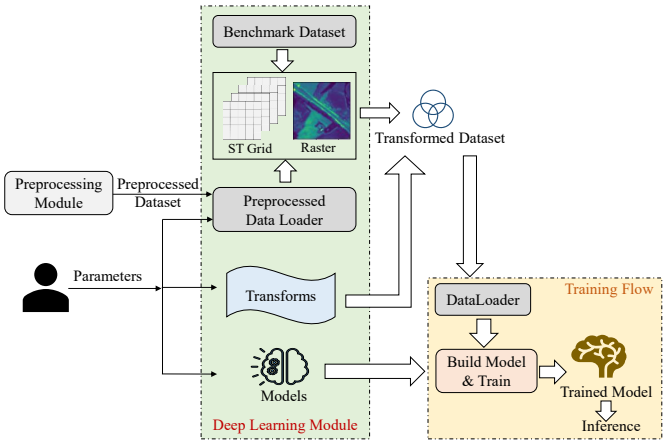


Fig. 4: GeoTorchAI Deep Learning Module

GeoTorchAI datasets module provides classes that allow defining any custom datasets instead of relying only on 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 III-B. Even if a dataset is not transformed or processed using our preprocessing module, it can still be loaded with GeoTorchAI. In the case of raster datasets, users can select the spectral bands they want to include in the feature vector. Besides, GeoTorchAI raster datasets provide the flexibility of including additional feature vectors extracted from raster images so that these datasets can be used to train models such as DeepSat [13] and Deepsat V2 [14], as discussed in Section II-C. For those users who lack domain knowledge on these spectral features of satellite images, our raster datasets support automatically extracting some commonly used spectral features and including them in the feature vector. Listing 1 shows the usage of a raster dataset. Setting the parameter $include\_additional\_features$ to $True$ in line 4 indicates that additional raster image features will automatically be extracted and included in the feature vector. These additional features are described in Section III-B2.

```
from geotorchai.datasets.raster import EuroSAT                          1
                                                                        2
eurosat_data = EuroSAT(root="data_path",                                3
                       include_additional_features=True)                4
inputs, labels, features = eurosat_data[0]                              5
print(inputs.shape, labels.shape, features.shape)                       6
```

Listing 1: Defining a GeoTorchAI Raster Dataset

Our spatiotemporal datasets provide three flexible options for retrieving the samples during an iteration, following the representations discussed in Section II-B. Firstly data samples can be iterated using the basic representation as shown in Listing 2. Setting the parameter $lead\_time$ to 24 in line 3 indicates that y_data will be at 24^{th} timestep after the x_data.

```
from geotorchai.datasets.grid import Temperature                        1
                                                                        2
weather_data = Temperature(root="data_path", lead_time=24)              3
x_data, y_data = weather_data[0]                                        4
print(x_data.shape, y_data.shape)                                       5
```

Listing 2: Basic Representation of a Grid-Based Dataset

Secondly, samples can also be retrieved as sequences of history and prediction following the sequential representation to enable the training of ConvLSTM [28] and other sequence models. Listing 3 shows the usage of such a representation in GeoTorchAI. The only difference with Listing 2 is the addition of line 4. The parameters $history\_length$ and $prediction\_length$ in line 4 indicate that x_data will consist of 48 timesteps, and y_data will consist of 24 timesteps immediately following the timesteps of x_data.

```
from geotorchai.datasets.grid import Temperature                        1
                                                                        2
weather_data = Temperature(root="data_path")                            3
weather_data.set_sequential_representation(history_length=              4
                                    48, prediction_length=24)           5
x_data, y_data = weather_data[0]                                        6
print(x_data.shape, y_data.shape)                                       7
```

Listing 3: Sequential Representation of a Grid-Based Dataset

Thirdly, data samples can be accessed in terms of closeness, period, and trend features using the periodical representation as proposed in ST-ResNet [2] so that they can be used to train models such as ST-ResNet and DeepSTN+ [27]. The process of applying the representation in GeoTorchAI is outlined in Listing 4. Similarly to the Listing 3, line 4 sets the periodical representation with the lengths of the most recent, recent, and least recent observations, respectively.

```
from geotorchai.datasets.grid import Temperature                        1
                                                                        2
weather_data = Temperature(root="data_path")                            3
weather_data.set_periodical_representation(len_closeness=3,             4
                                    len_period=4, len_trend=4)          5
data = weather_data[0]                                                  6
print(data["x_closeness"].shape, data["x_period"].shape,                7
          data["x_trend"].shape, data["y_data"].shape)                  8
```

Listing 4: Periodical Representation of a Grid-Based Dataset

Besides, each benchmark dataset provides the features and properties proposed in the corresponding dataset. For both categories of datasets (raster and grid-based), we minimize the number of public methods and mandatory parameters for ease of usage while providing some optional parameters so that datasets can be customized based on user preference. The benchmark datasets provided by the GeoTorchAI datasets module cover domains such as traffic forecasting, taxi and bike flow prediction, crowd volume prediction, raster image

classification, and raster image segmentation. Besides, it also contains various datasets for weather forecasting, which include but are not limited to the prediction of temperature, precipitation, geopotential, cloud coverage, and solar radiation.

2) GeoTorchAI Models: Similarly to the datasets module, the 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, 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.

Each model is implemented efficiently using existing PyTorch neural network layers as building blocks. As already discussed in the datasets module under Section III-A1, we minimize the number of public methods and mandatory parameters in each layer class. 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 training every batch, while cumulative training updates the weights once at the end of an epoch. Listings 5 and 6 show how to define a grid-based model and a raster model, respectively. Listing 5 shows the process of defining a model using the periodical representation in terms of most recent, recent, and distant features. Parameter $external\_dim = None$ in line 3 indicates that no external factors will be considered. Parameters $x\_closeness$, x_period, and x_trend in line 4 denote the most recent observations, near history, and distant history, respectively.

```
from geotorchai.models.grid import STResNet                    1
                                                               2
model = STResNet(external_dim=None)                            3
outputs = model(x_closeness, x_period, x_trend)                4
```

Listing 5: Defining a Sample Grid-Based Model

Listing 6 shows the definition of a raster classification model. The parameters in line 3 are set based on the target training dataset. Parameters $in\_channels$, in_height, in_width, and $num\_classes$ represent the number of channels or attributes, image height, image width, and the number of classes in the training dataset, respectively. The last parameter $num\_filtered\_features$ stands for the number of additional extracted features to be included in the feature vector. Parameters $inputs$ and $features$ in line 5 denote the images of a batch and additional extracted features of corresponding images.

```
from geotorchai.models.raster import DeepSatV2                 1
                                                               2
ml = DeepSatV2(in_channels=13, in_height=64, in_width=64,      3
               num_classes=10, num_filtered_features=13)       4
outputs = ml(inputs, features)                                 5
```

Listing 6: Defining a Sample Raster Model

Models available under the GeoTorchAI models module in the grid-based category can be applied to any forecasting task, such as weather forecasting, traffic flow forecasting, traffic volume forecasting, and crowd volume prediction. Besides, this module also contains models for raster imagery applications such as raster image classification and raster image segmentation. Some of the raster and grid-based spatiotemporal models included with GeoTorchAI are periodical CNN, ConvLSTM [28], ST-ResNet [2], DeepSTN+ [27], DeepSAT [13], DeepSAT-V2 [14], SatCNN [31], FCN [29], and UNet [30]. Among these models, Periodical CNN, ConvLSTM, ST-ResNet, and DeepSTN+ can be applied to spatiotemporal forecasting tasks, while DeepSAT, DeepSAT-V2, and SatCNN can be used to classify raster images. Models such as FCN and UNet are applicable to raster image segmentation tasks.

3) GeoTorchAI Transforms: GeoTorchAI transforms module provides transformation operations that can be applied to GeoTorchAI datasets during model training. Similar to Torchvision transformations, GeoTorchAI 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. GeoTorchAI raster transformation operations allow adding new feature vectors such as the normalized difference index of two bands as a new band to the raster image. As discussed earlier in Section II-C, these features improve the efficiency of modeling raster images. GeoTorchAI also allows transforming raster images offline before model training in a cluster computing environment, which we discuss later in GeoTorchAI preprocessing module under Section III-B. Listing 7 shows a sample transformation operation on GeoTorchAI. The selected transformation operation, appending normalized difference index of bands 1 and 2, can be applied to all sample images in the dataset during the training on the fly.

```
from geotorchai.transforms.raster import                       1
                            AppendNormalizedDifferenceIndex    2
from geotorchai.datasets.raster import EuroSAT                 3
                                                               4
append = AppendNormalizedDifferenceIndex(band_index1=1,        5
                                         band_index2=2)        6
train_data = EuroSAT(root="data_path", transform=append)       7
```

Listing 7: Defining a Sample Transformation Operation

B. Data Preprocessing Module

Raw spatiotemporal datasets require a number of preprocessing steps in order to convert them into spatiotemporal tensors described in Section II-A. Usually, these raw datasets are very large in size, and converting them into tensors requires a long processing time as well as high memory. For example, the TaxiNYC dataset used in STDN [1] has been prepared from New York City (NYC) taxi trip records data [39], which contains taxi trip information for every month since January 2009, and the size of each month's dataset is more than 2GB. 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. Besides spatiotemporal datasets, raster datasets also require preprocessing operations such as extracting raster image features, calculating normalized difference index, appending new bands to the image, deleting bands from the image, etc. Since raster image datasets are also huge in volume, processing raster images on distributed cluster computing systems can reduce the processing delay and memory-related errors. Performing the raster image 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.

Although the GeoTorchAI preprocessing module runs on Apache Sedona [15], users can use its methods in a proper Pythonic way. It has many Python functions for performing various types of processing and transformation operations on raster and spatiotemporal datasets. 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. Besides, the preprocessing module provides various Python methods, which internally use methods added to Apache Sedona to perform raster and spatiotemporal data processing. Along with the methods for reading and writing various spatial and non-spatial datasets as well as GeoTiff raster images, GeoTorchAI preprocessing module contains two sub-modules: one for spatiotemporal grid data preprocessing and the other for raster image preprocessing.

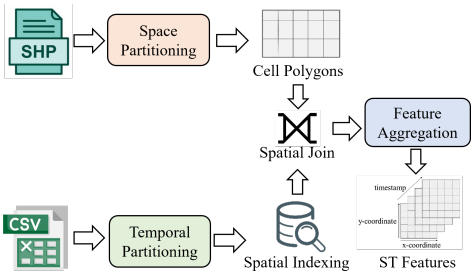


Fig. 5: Example Steps of Spatiotemporal Data Preprocessing

1) Spatiotemporal Data Preprocessing: Preprocessing of spatiotemporal data involves loading the dataset, converting the dataset into a spatiotemporal tensor, $\mathcal{X}^{grid}$ (details in Section II-A2), and writing the tensor into the disk for further usages, such as model training and testing. As discussed earlier in Section III-B, converting a spatiotemporal dataset into grid-based tensors includes preprocessing steps which include but are not limited to the partitioning of geographical space into a grid, calculating adjacency between geographical objects or grid cells, slicing the temporal range of the dataset into a number of time intervals, aggregating the feature vectors within each time interval for each grid cell. Available methods for spatiotemporal data processing in GeoTorchAI are implemented using efficient spatial joins on Apache Sedona as well as other joins and aggregation features of PySpark DataFrame. These spatial joins and other aggregation operations are a black box to the users of GeoTorchAI, and they can perform these operations by calling a minimum number of methods from the classes $SpacePartition$ and $STManager$ under the package *geotorchai.preprocessing.grid*. Figure 5 depicts some example steps of processing spatiotemporal non-imagery datasets. Besides forming trainable grid-based spatiotemporal tensors, the preprocessing module also supports the re-partitioning of grid-based spatial datasets by reducing the data volume with an end goal of reducing the training time [40].

Listing 8 shows how to convert a Spark DataFrame consisting of raw datasets into a DataFrame of spatiotemporal tensor format. Line 3 in the listing creates a spatial geometry-type column named $point$ from the columns lat and lon. The line 6 in the listing converts the geographical coverage of the dataset into a 12×16 grid and divides the total temporal range into various time intervals of length 30 minutes. The same method also aggregates the feature by counting the number of points covered by each temporal interval inside each grid cell. Method $get\_st\_grid\_dataframe$ also takes some optional parameters providing more controls to the tensor generation process. The details can be found in the API documentation of GeoTorchAI. The $DataFrame$ returned by the method $get\_st\_grid\_dataframe$ can be passed to the method $get\_st\_grid\_array$ under the class $STManager$ to further convert it into a $numpy$ array for the purpose of loading into PyTorch.

```
from geotorchai.preprocessing.grid import STManager as stm  1
                                                             2
spatial_df = stm.add_spatial_points(df=data_df, lat_column= 3
        "lat", lon_column="lon", new_column_alias="point") 4
                                                             5
st_df = stm.get_st_grid_dataframe(geo_df=spatial_df,         6
       geometry="point", partitions_x=12, partitions_y=16,   7
       col_date="time_column", step_duration_sec=1800)       8
```
Listing 8: Aggregating Features Within Grid Cells

2) Raster Data Preprocessing: Preprocessing operations performed on raster images 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 a band, inserting a new band, masking a band on an upper or lower threshold, etc. On the other hand, map algebra operations are used to extract features from raster images, such as getting different types of normalized difference indices, getting the mean, mode, modulus, and square root of a band, adding, subtracting, multiplying, and dividing bands, bitwise logical operations on bands, etc. Figure 6 depicts a few examples of raster image preprocessing. Besides these operations, our preprocessing module also allows extracting features related Gray-Level Co-occurrence Matrix (GLCM) of a raster image, such as GLCM contrast, GLCM dissimilarity, GLCM ASM, GLCM homogeneity, and

GLCM correlation. Raster image classification models such as DeepSAT-V2 recommend the inclusion of these features into the feature vector. The GLCM is a matrix that tracks how frequently various pixel brightness value combinations appear in an image [41]. GLCM features define various spatial relationships among pixels in an image [41].

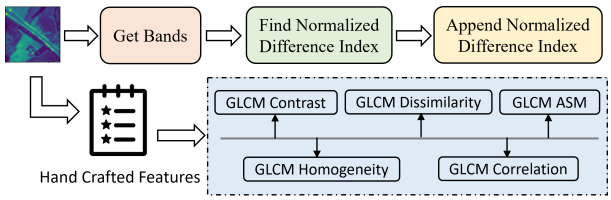


Fig. 6: Raster Image Preprocessing Example

Listing 9 shows an example of loading, transforming, and writing raster images. The method *load_geotiff_image* in line 5 can take some optional parameters to control the coordinate reference system of raster images during image loading. Similarly, the method *write_geotiff_image* in line 9 also takes optional parameters. We refer to the API documentation for the details.

```
import geotorchai.preprocessing as gpp                                  1
from geotorchai.preprocessing.raster import                             2
                                        RasterProcessing as rp          3
                                                                        4
rs_df = gpp.load_geotiff_image(path_to_dataset="path")                  5
appended_df = rp.append_normalized_difference_index(rs_df,              6
          band_index1=0, band_index2=1, column_data="data",             7
                                        column_n_bands="nBands")        8
gpp.write_geotiff_image(raster_df=appended_df,                          9
                                destination_path="new_path")            10
```

Listing 9: An Example of Raster Transformation

C. DFtoTorch Converter

The product of scalable preprocessing in Apache Sedona is a DataFrame, a structured form equivalent to a relational database table, composed of organized columns. Notably, DataFrames procured from Apache Sedona or Spark are not inherently suited for direct utilization as a PyTorch Dataset, which conventionally employs tensors. To adapt preprocessed DataFrames for PyTorch datasets, a straightforward but inefficient solution exists. The process begins by collecting the full preprocessed DataFrame onto the master node from all worker nodes. The DataFrame is then transformed into an array structure, subsequently stored on the disk. This stored array is imported into PyTorch and converted into the requisite tensor format. However, this approach presents considerable drawbacks. In scenarios of distributed training and preprocessing, the large volume nature of DataFrames becomes a logistical challenge, often exceeding the master node's memory capabilities. Additionally, the operations of gathering the extensive data and writing to files are time-intensive, potentially hampering the overall workflow efficiency.

Addressing the limitations previously outlined, we propose an additional component, *DFtoTorch Converter*, designed to facilitate the mapping of preprocessed DataFrames into PyTorch datasets. The structural design of this innovative module is illustrated in Figure 7. The *DFtoTorch Converter* consists of two primary segments: the *DF Formatter* and the *Row Transformer*. The *DF Formatter* takes the preprocessed DataFrame as input and efficiently maps each individual row into an array format, mirroring the eventual tensor configuration. This procedure is executed in a distributed manner, negating the need for centralized aggregation of the entire DataFrame on the master node. Importantly, the parameters of this mapping are influenced by the intended application domain, including classifications, segmentations, or spatiotemporal predictions, and it may incorporate additional parameters contingent on user requirements. However, the output of the mapping process is still a DataFrame, which is processed subsequently by the *Row Transformer* component to yield PyTorch tensors from each row in the DataFrame. This transformation leverages the Petastorm tool [42], executing user-specified transformations on the DataFrame rows and returning the rows as batches. Parameters guiding this phase comprise batch size, transformation operations, and application-specific variables, for instance, dimensions pertinent to raster imagery tasks. The transformation spec returned by this step is required during iterating over the converted dataset with PyTorch dataloaders.

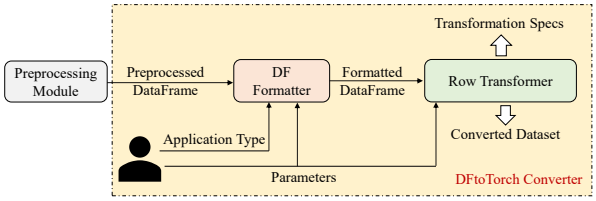


Fig. 7: Preprocessed DataFrame to PyTorch dataset converter

IV. Design Principles

GeoTorchAI is designed in such a way that it has the necessary building blocks for developing raster and spatiotemporal applications within the PyTorch ecosystem. Various functionalities available in GeoTorchAI deep learning module are compatible with PyTorch core units, such as neural network layers, datasets, and transformations. We make the deep learning module of GeoTorchAI GPU compatible so that PyTorch-provided scalability and parallelism on GPU can be achieved with GPU-configured devices.

Although the data preprocessing module has dependencies on external big data processing libraries such as PySpark and Apache Sedona, the deep learning module only depends on PyTorch. Since the datasets component of the deep learning module provides preprocessed and trainable state-of-the-art benchmark datasets, designing applications with such benchmark datasets can be completed without requiring big data-related dependencies. Furthermore, to help machine learning practitioners build raster and spatiotemporal applications with their preferred raw datasets, our preprocessing module enables raster and spatiotemporal data processing in a pure Pythonic

way without requiring the coding knowledge of Apache Spark, Apache Sedona, and other big data processing libraries while providing the scalability of Apache Spark at the same time.

Our preprocessing module is designed such that it minimizes the number of methods and classes in the API. Users can perform end-to-end spatiotemporal data preprocessing, which starts by loading raw datasets and ends by generating a trainable Tensor-shaped array, with a minimum number of method calls. It helps the users understand the API fast and reduces their confusion.

The source code of GeoTorchAI is publicly available on GitHub [1]. The repository is ready to accept contributions from external contributors with novel feature recommendations. In order to ensure the quality and proper readability of the codebase, we apply standard software development practices while writing code for all components of GeoTorchAI. The documentation on the API usage has been made available online and is available on the GitHub repository. The API usage guide documents all classes and methods thoroughly with examples. The documentation also provides end-to-end code examples and tutorials for all types of use cases to ease the adoption of GeoTorchAI to related applications. Unit tests have been added to test the methods of various components properly. The library can be deployed on platforms such as Linux, Windows, and macOS.

V. Experimental Evaluation

Using some of our benchmark datasets, we assess the performance of the deep learning module of our proposed framework, GeoTorchAI, on raster and spatiotemporal models. We also compare the models in terms of the training time and evaluate the impact of several input parameters on training time in both CPU-based and GPU-based training. In addition, we check the effectiveness of the preprocessing module on tasks such as raster processing and spatiotemporal grid-based tensor preparation from raw datasets.

A. Experimental Settings

1) System Configuration: On a machine with an Intel NVIDIA GPU (GM107GL [Quadro K2200]) with 640 CUDA cores, 120 GB of RAM, and a 4 TB hard drive, we conduct all the model training experiments. While the majority of the experiments are run with GPU enabled, we also run a few on the CPU to compare runtimes between the two.

2) Datasets: GeoTorchAI benchmark datasets used by grid-based spatiotemporal applications are listed in Table II, while Table III lists the raster imagery datasets. In addition to these datasets, GeoTorchAI contains five different ready-to-use weather forecasting datasets, which include forecasting of temperature, geopotential, total precipitation, total cloud cover, and total incident solar radiation [43]. All of these datasets contain weather data for 2018 in grid shape 5.625 deg vs. 2.8125 deg, but users have the flexibility of changing these parameters according to their needs. The exact shape of the grid in 5.625 deg vs. 2.8125 deg is 32×64. Weather records in every dataset are separated by one-hour time intervals.

We perform the experimental evaluation of the deep learning module on four types of applications: spatiotemporal traffic prediction task, weather forecasting task, satellite image classification task, and satellite image segmentation task. For the spatiotemporal prediction task, we use three datasets from GeoTorchAI benchmark datasets: BikeNYC-DeepSTN [27], TaxiBJ21 [44], and YellowTrip-NYC dataset prepared with GeoTorchAI preprocessing module. We prepare a spatiotemporal tensor from NYC yellow trip records [39] and use the prepared tensor to train grid-based spatiotemporal models on traffic forecasting applications. The details of this dataset are available in Table II. The advantage of this dataset is that it supports all spatiotemporal tensor representations and can be used to train any spatiotemporal prediction model, a feature missing in all other NYC-based datasets. We evaluate the weather forecasting task with three datasets (temperature, total precipitation, and total cloud cover) out of five weather forecasting datasets described in the previous paragraph. On the other hand, for the task of satellite image classification, we use the datasets EuroSAT [3] and SlumDetection [45]. Satellite image segmentation is performed with 38-Cloud dataset [4].

3) Evaluation Metrics: We evaluate the predictive performance of spatiotemporal models with two metrics: Mean Absolute Error (MAE) and Root Mean Square Error. For the satellite image classification and segmentation tasks, we measure the performance in terms of classification and segmentation accuracy. In the case of evaluating the preprocessing module, the performance is measured against the elapsed time.

B. Evaluating Spatiotemporal Tensor Preparation

In order to evaluate the scalability of the preprocessing module in terms of preparing a grid-based spatiotemporal tensor, we compare the elapsed time and memory consumption with the GeoPandas library [46]. It should be noted that we perform the experiments in a single machine instead of a distributed environment for a fair comparison with the GeoPandas library because GeoPandas cannot run in a distributed setting. As the evaluation dataset in this experiment, we pick the NYC tax trip data [39] and take four versions of this dataset varying the size. The number of trip records in these four datasets are 1.4 million, 14 million, 100 million, and 250 million, respectively. The taxi trip dataset of a single month contains approximately 14 million records. For the smallest dataset with 1.4 million samples, we perform a spatially stratified sampling on a single month's dataset. We prepare the larger datasets by merging the records from several consecutive months. Figure 9 depicts the scalability of our preprocessing module in terms of preparing a grid-based spatiotemporal tensor.

The depicted figure shows that the GeoTorchAI preprocessing module outperforms the GeoPandas library by order of magnitude in terms of both memory usage and elapsed time. Memory usage by the GeoPandas library increases significantly as the data size increases and results in an out-of-memory error for the largest dataset, while GeoTorchAI

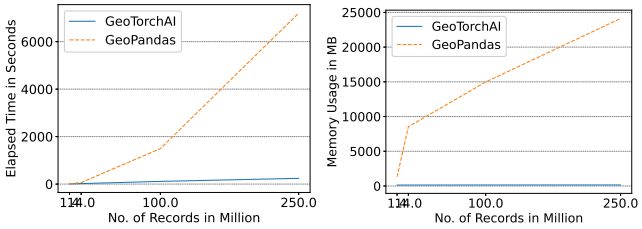
[1]https://github.com/wherobots/GeoTorchAI

TABLE II: Grid-Based Spatiotemporal Datasets

Dataset	Data Type	Grid Shape	Time Interval	Time Duration
BikeNYC-DeepSTN[27]	Bike Flow	21×12	1 Hour	01/04/2014 - 30/09/2014
TaxiNYC-STDN[1]	Taxi Flow and Volume	10×20	30 Minutes	01/01/2015 - 01/03/2015
BikeNYC-STDN[1]	Bike Flow and Volume	10×20	30 Minutes	01/07/2016 - 29/08/2016
TaxiBJ21[44]	Taxi Flow	32×32	30 Minutes	November 2012, November 2014, and November 2015
YellowTrip-NYC	Taxi Pickup and Dropoff	12×16	30 Minutes	01/10/2010 - 31/12/2010

TABLE III: Raster Image Datasets

Dataset	Type	Image Shape	Classes	Bands
SAT-6[13]	Multi-class Classification	28×28	6	4
SAT-4[13]	Multi-class Classification	28×28	4	4
EuroSAT[3]	Multi-class Classification	64×64	10	13
SlumDetection[45]	Binary Classification	32×32	2	4
38-Cloud[4]	Segmentation	384×384	-	4



(a) ComparingElapsed Time (b) Comparing Memory Usage

Fig. 8: Elapsed Time and Memory Usage During Grid-Based Spatiotemporal Tensor Preparation

remains consistent in terms of memory for all datasets. The processing time in the GeoPandas library is also much higher compared to that of GeoTorchAI. GeoPnadas library cannot finish the processing for the largest dataset and results in memory error after 2.5 hours of processing.

C. Evaluating Spatiotemporal Traffic Prediction Task

For the BikeNYC-DeepSTN dataset [27], the task is to predict the flow of the bikes in New York City at a future time interval, given the historical bike-flow records of a fixed number of previous time intervals. Similarly, in the case of TaxiBJ21 dataset [44], the task is to predict the flow of taxis in Beijing City. In the case of the YellowTrip-NYC dataset formed with the GeoTorchAI preprocessing module and released by us, we predict the number of taxi pickups and dropoffs at various cells in New York City. We split each of these datasets into three parts: train, validation, and test sets. The training dataset consists of the records for the first 80% of time periods. Out of the remaining 20% records, the first half of the records denote the validation dataset, while the test dataset contains the last half. Training and validation datasets are used during the training process, and we evaluate all models on the test dataset. Mean squared error loss and Adam optimizer are used as the loss function and optimization function, respectively. We initialize and train each model for five iterations, while each iteration consists of epochs. The number of epochs is not fixed for all models because different models show convergence after different numbers of epochs. We train each model until MSE or RMSE continues to reduce on the validation dataset. We use early stopping criteria to stop the training early if the performance on the validation dataset does not improve after a number of epochs. We record the MAE and RMSE obtained by the trained models in each of the 5 iterations and report the average of the results as well as the variation from the average. We follow the incremental training procedure where loss is back-propagated, and model weights are updated after every batch of data.

TABLE IV: Traffic Prediction with Spatiotemporal Models

Datasets	Metric	Models			
		Periodical CNN	ConvLSTM	ST-ResNet	DeepSTN+
BikeNYC-DeepSTN	MAE	6.032±0.317	4.655±0.116	2.912±0.090	2.325±0.056
	RMSE	14.495±0.351	11.832±0.254	7.317±0.227	6.085±0.133
TaxiBJ21	MAE	0.048±0.005	0.082±0.006	0.044±0.004	0.019±0.003
	RMSE	0.087±0.008	0.120±0.009	0.073±0.005	0.032±0.006
YellowTrip-NYC	MAE	0.626±0.082	0.068±0.009	0.061±0.010	0.0008±0.0001
	RMSE	0.810±0.066	0.164±0.016	0.138±0.056	0.034±0.007

Table IV reports the prediction errors of various models on various traffic datasets. Reported results indicate that although all models perform very close to each other on TaxiBJ21 [44] dataset, model DeepSTN+ [27] outperforms other models significantly on the BikeNYC-DeepSTN dataset and YellowTrip-NYC dataset. It also proves the correctness of the YellowTrip-NYC dataset prepared using the preprocessing module because models that show high performance on other datasets also perform well on this dataset. The superior performance of DeepSTN+ and ST-ResNet over the ConvLSTM model validates the usefulness of closeness, period, and trend features proposed by ST-ResNet [2] model. Also, the performance of a model changes very slightly from one run to another run which proves the consistency of a model trained using GeoTorchAI.

D. Evaluating Spatiotemporal Weather Forecasting Task

The approach followed for evaluating weather forecasting tasks is similar to the approach used for the evaluation of traffic prediction applications. We evaluate the weather forecasting performance of four models previously used for evaluating traffic prediction in Section V-C with three weather datasets - temperature, total precipitation, and total cloud cover. Hyperparameter settings are similar to those mentioned in Section V-C, and datasets are distributed into train, validation, and test sets in a similar way. Another similarity with the evaluation of traffic prediction is that we train each model for 5 iterations (each iteration with multiple epochs) and report the average MAE and RMSE along with the maximum and minimum variations from the average.

TABLE V: Weather Forecasting with Spatiotemporal Models

Datasets	Metric	Models			
		Periodical CNN	ConvLSTM	ST-ResNet	DeepSTN+
Temperature	MAE	2.398±0.503	0.181±0.037	1.284±0.245	0.168±0.044
	RMSE	3.361±0.318	0.288±0.055	2.852±0.187	0.274±0.029
Total Precipitation	MAE	0.00014± ≈ 0	0.000059± ≈ 0	0.00018± ≈ 0	0.000051± ≈ 0
	RMSE	0.00040± ≈ 0	0.00032± ≈ 0	0.0016± ≈ 0	0.00020± ≈ 0
Total Cloud Cover	MAE	0.265±0.164	0.072±0.004	0.098259±0.008	0.060±0.006
	RMSE	0.320±0.093	0.135±0.026	0.176±0.088	0.105±0.061

Table V reports the mean absolute error and root mean square error of spatiotemporal models on various weather forecasting datasets. Similarly to the traffic prediction applications, model DeepSTN+ outperforms all other models on weather forecasting tasks also. It should be noted that the model ConvLSTM performs almost similarly to the DeepSTN+ model and outperforms other models on all weather forecasting datasets. This is due to the fact that the ConvLSTM model is specially designed for weather forecasting applications which are less impacted by closeness, period, and trend factors compared to the traffic prediction tasks.

E. Evaluating Raster Classification and Segmentation Tasks

We experiment with the raster image classification task on two datasets: EuroSAT [3] and SAT6 [13], while 38-Cloud [4] dataset is used for evaluating the segmentation task. Those datasets that do not have training and test sets predefined, such as EuroSAT, are divided into 80% training set and 20% test set, and model evaluation is performed on the test dataset. Models DeepSAT V2 [14] and SatCNN [31] are used for experimenting with the classification task, while the segmentation task is evaluated with UNet++ [47], UNet [30] and Fully Convolutional Network (FCN) [29] models. We utilize Cross Entropy Loss as the loss function for all models while keeping the other parameters and the number of training iterations the same as those used for traffic prediction and weather forecasting tasks.

From each dataset, we extract six textural features for classification with DeepSAT V2 utilizing the authors' proposal of the handcrafted features. Additionally, we extract seven spectral features from the EuroSAT dataset as well as three spectral characteristics from the SAT6 dataset. Different spectral indices suggested in the literature, such as the Normalized Density Vegetation Index (NDVI), the Normalized Density Water Index (NDWI), etc., are examples of spectral features. Because the SAT6 dataset lacks the short-wave infrared band, which is necessary for many spectral indices, we are unable to extract all the spectral indices from this dataset. Textural features retrieved from each dataset include contrast, dissimilarity, correlation, homogeneity, momentum, and energy. All the bands from each dataset listed in Table III are included in the feature vectors. Table VI reports the classification and segmentation accuracy of evaluated models on various datasets. According to the reported results, the performance of the tested models on both datasets is relatively comparable, and none of the models can completely dominate all datasets. Even though DeepSAT V2 [14] has fewer convolution layers than SatCNN [31], it performs equally as well, demonstrating the success of the feature fusion concept and the custom features that DeepSAT V2 has put forth.

TABLE VI: Accuracy of Raster Models on Satellite Image Classification and Segmentation

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%
UNet++	38-Cloud	Segmentation	98.490±0.407%

F. Evaluating Training Time

To compare the deep learning models in terms of the training time, we record the average training time of all the models for an epoch and report it in Table VII. For the grid-based spatiotemporal models, such as Periodical CNN, ConvLSTM, ST-ResNet, and DeepSTN+, we report the training time for training with the Temperature dataset. Training time reported for the raster image classification models such as DeepSAT V2 and SatCNN includes the training with EuroSAT images. In the case of raster image segmentation models, including Fully Convolutional Network, UNet, and UNet++, we report the training time for training with the 38-Cloud dataset.

TABLE VII: Training Time of Various Models for a Single Epoch

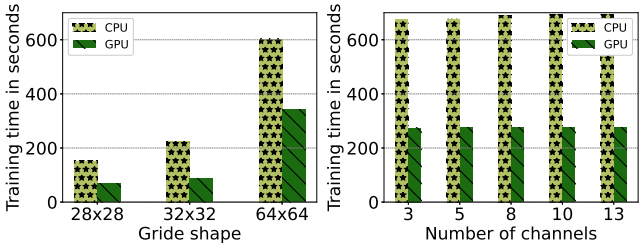
Dataset	Application	Model	Training Time Per Epoch
Temperature	Prediction	Periodical CNN	2.868 Seconds
		ConvLSTM	588.094 Seconds
		ST-ResNet	27.416 Seconds
		DeepSTN+	20.775 Seconds
EuroSAT	Classification	DeepSAT V2	172.570 seconds
		SatCNN	897.383 seconds
38-Cloud	Segmentation	FCN	1435.495 Seconds
		UNet	1765.574 Seconds
		UNet++	2139.914 Seconds

Based on the results reported in Table VII, it is evident that ConvLSTM is the slowest running model, although it is not the best-performing model in terms of MAE and RMSE according to the results of the prediction reported in Tables IV and V. The prediction model with the lowest error, DeepSTN+, runs much faster than the ConvLSTM model. Regarding the raster classification models, DeepSAT V2 runs more than 5 times faster while both models have almost similar classification accuracy according to Table VI. We observe a variation in this trend in the case of raster segmentation models. UNet++ is the slowest among the three segmentation models, and it is the best model in terms of segmentation accuracy according to Table VI. Analyzing the training time and performance of prediction, classification, and segmentation models, we can conclude that the performance of a model is not directly related to the training time. Usually, models consisting of lots of neural network layers and parameters run very slowly, but these models with many layers and parameters may not always have the lowest prediction error or highest classification/segmentation

accuracy, and faster-running models with simple architecture might outperform them.

G. Evaluating the Impact of Grid Shape and Number of Bands

To evaluate the impact of grid shape and number of bands, we vary the number of bands and grid size and measure the training time of a single epoch. The training time is measured by running the models separately on CPU and GPU. The model SatCNN is trained with the EuroSAT dataset for a single epoch by setting all the hyperparameters as described in Section V-E. We train the model on 3, 5, 8, 10, and 13 bands, respectively, to evaluate the impact of the number of bands on the training time. We use three grids for the evaluation of the grid size: 28×28, 32×32, and 64×64. During the experiments on the grid size, bands are fixed to have three RGB bands.



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

Fig. 9: Time to Run an Epoch on Varying Number of Channels and Grid Shapes

Figure 9 shows how variations in the number of bands and grid size affect how long it takes to train an epoch. The findings allow us to draw the conclusion that while grid shape has a considerable influence on training time, the number of channels or bands has no discernible effect. Additionally, running a model on a GPU rather than a CPU can significantly reduce the training time.

H. Evaluating the Impact of Raster Preprocessing

By employing the preprocessing module to do the raster transformation operations before model training rather than doing so on the fly while training, we assess the effectiveness of our preprocessing module. When doing raster preprocessing and model training separately, we record the total amount of time spent on preprocessing for various counts of transformation operations, as well as the amount of time spent on training the model using the modified images. In addition, we also keep track of how long it takes to train a model while performing similar transformations during training. We repeat the procedure for transformation counts of 1, 2, 3, 4, and 5. The goal of each transformation operation is to append a normalized difference index to the image data.

Table VIII reports the data preprocessing and model training time in various settings. The effectiveness of our preprocessing module is demonstrated by the fact that the sum of pre-transformation time and model training time with pre-transformed data is significantly less than the training time with transformation on the fly. Although preprocessing time can be further decreased by executing the preprocessing module in a clustered environment, we execute the preprocessing on a single machine in order to make a fair comparison. Data loading, data transformation, and data writing are all included in the preprocessing time that is shown in Table VIII. The writing operation, which is done once at the end of all transformations, dominates preprocessing time, resulting in a trivial change in the preprocessing time along with an increase in the number of transformations. Because no transformation occurs during training with pre-transformed data, the training time does not rise for a higher number of transformations. The efficiency of pre-transformation with our preprocessing module is proven by the fact that training time grows as the number of transformations increases when transformations happen during the training.

TABLE VIII: Elapsed Time in Seconds for Various Training and Preprocessing Settings

Transforms 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

VI. Conclusion

In this study, we introduce GeoTorchAI, a spatiotemporal deep-learning framework designed specifically for raster imagery and grid-based non-imagery datasets. Emulating PyTorch's structure, GeoTorchAI extends its foundational classes to provide support for neural networks, datasets, and transformations, facilitating advanced deep learning applications on grid-based spatiotemporal datasets and satellite image datasets that have not been covered by existing deep learning frameworks. Additionally, GeoTorchAI's preprocessing module 1) supports training and testing models using raw spatiotemporal datasets rather than being limited to only ready-to-use benchmark datasets, and 2) supports scalable data preprocessing and transformation, both of which further contribute to a significant reduction in training time. Finally, *DFtoTorch Converter* in GeoTorchAI can map processed DataFrames into PyTorch tensors. We empirically assess the satellite image classification and segmentation tasks, spatiotemporal traffic prediction and weather forecasting tasks, scalability of the preprocessing module, as well as how different input parameters affect training time on both CPU and GPU. Scalable data preparation prior to model training is effective, as shown by our empirical evaluation of the preprocessing module.

By consistently incorporating new preprocessing functions, transforms, benchmark datasets, and cutting-edge models, we seek to improve the GeoTorchAI features. In the domains of raster images and grid-based datasets, we are working on incorporating new datasets and models from a more diverse variety of domain applications.

REFERENCES

[1] H. Yao, X. Tang, H. Wei, G. Zheng, and Z. J. Li, "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction," in *AAAI*, 2019.

[2] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, ser. AAAI'17. AAAI Press, 2017, p. 1655–1661.

[3] P. Helber, B. Bischke, A. Dengel, and D. Borth, "Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019.

[4] S. Mohajerani and P. Saeedi, "Cloud-net: An end-to-end cloud detection algorithm for landsat 8 imagery," 07 2019, pp. 1029–1032.

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[7] F. Chollet *et al.* (2015) Keras. [Online]. Available: https://github.com/fchollet/keras

[8] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *ArXiv*, vol. abs/1512.01274, 2015.

[9] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1485–1488. [Online]. Available: https://doi.org/10.1145/1873951.1874254

[10] R. Jiang, D. Yin, Z. Wang, Y. Wang, J. Deng, H. Liu, Z. Cai, J. Deng, X. Song, and R. Shibasaki, *DL-Traff: Survey and Benchmark of Deep Learning Models for Urban Traffic Prediction*. New York, NY, USA: Association for Computing Machinery, 2021, p. 4515–4525. [Online]. Available: https://doi.org/10.1145/3459637.3482000

[11] B. Rozemberczki, P. Scherer, Y. He, G. Panagopoulos, A. Riedel, M. Astefanoaei, O. Kiss, F. Beres, G. López, N. Collignon, and R. Sarkar, *PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models*. New York, NY, USA: Association for Computing Machinery, 2021, p. 4564–4573. [Online]. Available: https://doi.org/10.1145/3459637.3482014

[12] P. Goyal, S. R. Chhetri, N. Mehrabi, E. Ferrara, and A. Canedo, "Dynamicgem: A library for dynamic graph embedding methods," *ArXiv*, vol. abs/1811.10734, 2018.

[13] S. Basu, S. Ganguly, S. Mukhopadhyay, R. DiBiano, M. Karki, and R. Nemani, "Deepsat: A learning framework for satellite imagery," in *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. SIGSPATIAL '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2820783.2820816

[14] Q. Liu, S. Basu, S. Ganguly, S. Mukhopadhyay, R. DiBiano, M. Karki, and R. R. Nemani, "Deepsat v2: feature augmented convolutional neural nets for satellite image classification," *Remote Sensing Letters*, vol. 11, pp. 156 – 165, 2019.

[15] (2020) Apache sedona (incubating). [Online]. Available: https://sedona.apache.org/

[16] A. J. Stewart, C. Robinson, I. A. Corley, A. Ortiz, J. M. L. Ferres, and A. Banerjee, "Torchgeo: Deep learning with geospatial data," ser. SIGSPATIAL '22, 2022.

[17] K. Chowdhury and M. Sarwat, "Geotorch: A spatiotemporal deep learning framework," ser. SIGSPATIAL '22, 2022.

[18] ——, "A demonstration of geotorchai: A spatiotemporal deep learning framework," in *Companion of the 2023 International Conference on Management of Data*, ser. SIGMOD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 195–198. [Online]. Available: https://doi.org/10.1145/3555041.3589734

[19] S. Bhattacharya, C. Braun, and U. Leopold, "A tensor based framework for large scale spatio-temporal raster data processing," *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLII-4/W14, pp. 3–9, 08 2019.

[20] H. Yao, F. Wu, J. Ke, X. Tang, Y. Jia, S. Lu, P. Gong, J. Ye, and Z. J. Li, "Deep multi-view spatial-temporal network for taxi demand prediction," in *AAAI*, 2018.

[21] G. Lai, W.-C. Chang, Y. Yang, and H. Liu, "Modeling long- and short-term temporal patterns with deep neural networks," *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, 2018.

[22] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "Geoman: Multi-level attention networks for geo-sensory time series prediction," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 3428–3434.

[23] S.-Y. Shih, F.-K. Sun, and H.-y. Lee, "Temporal pattern attention for multivariate time series forecasting," *Mach. Learn.*, vol. 108, no. 8–9, p. 1421–1441, sep 2019. [Online]. Available: https://doi.org/10.1007/s10994-019-05815-0

[24] Z. Pan, Y. Liang, W. Wang, Y. Yu, Y. Zheng, and J. Zhang, "Urban traffic prediction from spatio-temporal data using deep meta learning," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1720–1730. [Online]. Available: https://doi.org/10.1145/3292500.3330884

[25] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan, *Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting*. Red Hook, NY, USA: Curran Associates Inc., 2019.

[26] C. Zheng, X. Fan, C. Wang, and J. Qi, "Gman: A graph multi-attention network for traffic prediction," in *AAAI*, 2020.

[27] Z. Lin, J. Feng, Z. Lu, Y. Li, and D. Jin, "Deepstn+: Context-aware spatial-temporal neural network for crowd flow prediction in metropolis," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 1020–1027, 07 2019.

[28] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 802–810.

[29] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 640–651, 2017.

[30] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, Eds. Cham: Springer International Publishing, 2015, pp. 234–241.

[31] Y. Zhong, F. Fei, Y. Liu, B. Zhao, H. Jiao, and L. Zhang, "Satcnn: satellite image dataset classification using agile convolutional neural networks," *Remote Sensing Letters*, vol. 8, no. 2, pp. 136–145, 2017. [Online]. Available: https://doi.org/10.1080/2150704X.2016.1235299

[32] S. Basu, M. Karki, S. Mukhopadhyay, R. Dibiano, s. ganguly, R. Nemani, and S. Gayaka, "Deep neural networks for texture classification—a theoretical analysis," *Neural Networks*, vol. 97, pp. 173–182, 01 2018.

[33] P. Scherer and P. Lio', "Learning distributed representations of graphs with geo2dr," *ArXiv*, vol. abs/2003.05926, 2020.

[34] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *ArXiv*, vol. abs/1903.02428, 2019.

[35] J. Hu, S. Qian, Q. Fang, Y. Wang, Q. Zhao, H. Zhang, and C. Xu, "Efficient graph deep learning in tensorflow with tf_geometric," *Proceedings of the 29th ACM International Conference on Multimedia*, 2021.

[36] P. Goyal and E. Ferrara, "Gem: A python package for graph embedding methods," *Journal of Open Source Software*, vol. 3, p. 876, 09 2018.

[37] C. Data61, "Stellargraph machine learning library," https://github.com/stellargraph/stellargraph, 2018.

[38] D. Grattarola and C. Alippi, "Graph neural networks in tensorflow and keras with spektral [application notes]," *Comp. Intell. Mag.*, vol. 16, no. 1, p. 99–106, feb 2021. [Online]. Available: https://doi.org/10.1109/MCI.2020.3039072

[39] (2009) Tlc trip record data. [Online]. Available: https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page

[40] K. Chowdhury, V. V. Meduri, and M. Sarwat, "A machine learning-aware data re-partitioning framework for spatial datasets," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 2426–2439.

[41] M. Hall-Beyer, "Glcm texture: A tutorial v. 3.0 march 2017," 2017.

[42] R. Gruener, O. Cheng, and Y. Litvin, "Introducing petastorm: Uber atg's data access library for deep learning," *URL: https://eng.uber.com/petastorm/*, 2018.

[43] S. Rasp, P. D. Dueben, S. Scher, J. A. Weyn, S. Mouatadid, and N. Thuerey, "Weatherbench: A benchmark data set for data-driven weather forecasting," *Journal of Advances in Modeling Earth Systems*, vol. 12, 2020.

[44] W. Jiang, "Taxibj21 : An open crowd flow dataset based on beijing taxi gps trajectories," *Internet Technology Letters*, 04 2021.

[45] F. Baylé. (2017) Slum and informal settlements detection. [Online]. Available: https://www.kaggle.com/fedebayle/slums-argentina

[46] K. Jordahl, "Geopandas: Python tools for geographic data," *URL: https://github. com/geopandas/geopandas*, 2014.

[47] Z. Zhou, M. M. Rahman Siddiquee, N. Tajbakhsh, and J. Liang, "Unet++: A nested u-net architecture for medical image segmentation," in *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, D. Stoyanov, Z. Taylor, G. Carneiro, T. Syeda-Mahmood, A. Martel, L. Maier-Hein, J. M. R. Tavares, A. Bradley, J. P. Papa, V. Belagiannis, J. C. Nascimento, Z. Lu, S. Conjeti, M. Moradi, H. Greenspan, and A. Madabhushi, Eds. Springer International Publishing, 2018, pp. 3–11.