TORCHGEO: DEEP LEARNING WITH GEOSPATIAL DATA

Anonymous authors
Paper under double-blind review

ABSTRACT

Remotely sensed geospatial data are critical for earth observation applications including precision agriculture, urban planning, disaster monitoring and response, and climate change research, among others. Deep learning methods are particularly promising for modeling many earth observation tasks given the success of deep neural networks in similar computer vision tasks and the sheer volume of remotely sensed imagery available. However, the variance in data collection methods and handling of geospatial metadata make the application of deep learning methodology to remotely sensed data nontrivial. For example, satellite imagery often includes additional spectral bands beyond red, green, and blue and must be joined to other geospatial data sources that can have differing coordinate systems, bounds, and resolutions. To help realize the potential of deep learning for remote sensing applications, we introduce TorchGeo, a Python library for integrating geospatial data into the PyTorch deep learning ecosystem. TorchGeo provides data loaders for a variety of benchmark datasets, composable datasets for generic geospatial data sources, samplers for geospatial data, and transforms that work with multispectral imagery. TorchGeo is also the first library to provide pre-trained models for multispectral satellite imagery, allowing for advances in transfer learning on downstream earth observation tasks with limited labeled data. We use TorchGeo to create reproducible benchmark results on existing datasets, benchmark our proposed method for preprocessing geospatial imagery on-the-fly, and investigate the differences between ImageNet pre-training and in-domain self-supervised pre-training on model performance across several datasets. We aim for TorchGeo to become a new standard for reproducibility and for driving progress at the intersection of deep learning and remotely sensed geospatial data.

1 INTRODUCTION

With the explosion in availability of satellite and aerial imagery over the past decades, there has been increasing interest in the use of remotely sensed imagery in earth observation (EO) applications. These applications range from precision agriculture [31] and forestry [26], to natural and man-made disaster monitoring [53], to weather and climate change [44]. At the same time, advancements in machine learning (ML), larger curated benchmark datasets, and increased compute power, have led to great successes in domains like computer vision (CV), natural language processing (NLP), and audio processing. However, the wide-spread success and popularity of machine learning—particularly of deep learning methods—in these domains has not fully transferred to the EO domain, despite the existence of petabytes of freely available satellite imagery and a variety of benchmark datasets for different EO tasks. This is not to say that there are not successful applications of ML in EO, but that the full potential of the intersection of these fields has not been reached. Indeed, a recent book by Camps-Valls et al. [5] thoroughly details work at the intersection of deep learning, geospatial data, and the earth sciences. Increasing amounts of research on self-supervised and unsupervised learning methods specific to remotely sensed geospatial imagery [1; 22; 27] bring the promise of developing *generic* models that can be tuned to various downstream EO tasks. Recent large scale efforts, such as the creation of a global 10 m resolution land cover map [24] or the creation of global 30 m forest maps [43], pair the huge amount of available remotely sensed imagery with modern GPU accelerated models. To reach the full joint potential of these fields, we believe that we need tools for facilitating research and managing the complexities of both geospatial data and modern machine learning pipelines. We describe the challenges of this below, and detail our proposed solution, TorchGeo.

One major challenge in many EO tasks is the large amount of diversity in the content of geospatial imagery datasets compared to datasets collected for traditional vision applications. For example, most conventional cameras capture 3-channel RGB imagery, however most satellite platforms capture *different*

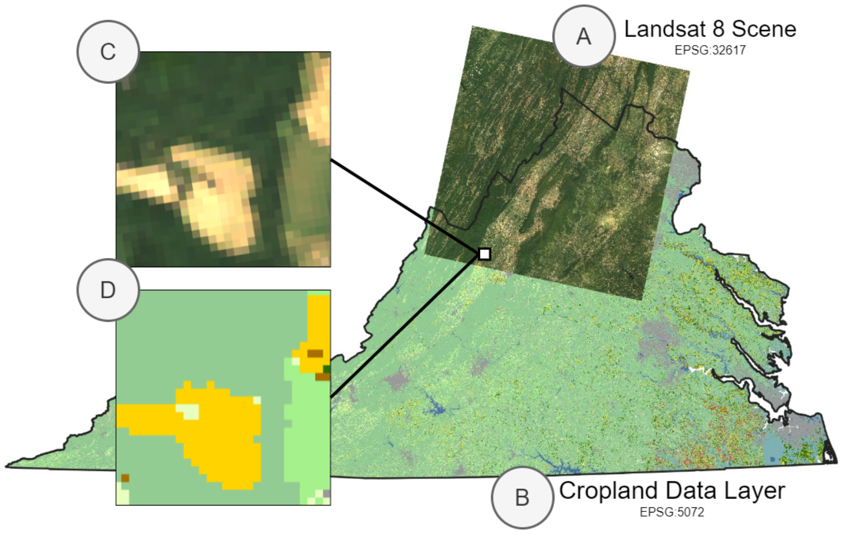

Figure 1: An illustration of the challenges in sampling from heterogeneous geospatial data layers. (**A** and **B**) show example geospatial data layers that a user may want to sample *pixel-aligned* data from. As these layers have differing coordinate reference systems, patches of imagery (**C** and **D**) sampled from these layers *that cover the same area* will not be pixel-aligned. TorchGeo transparently performs the appropriate alignment steps (reprojecting and resampling) during data loading such that users can train deep learning models without having to manually align the data layers.

sets of spectral bands. The Landsat 8 satellite [45] collects 11 bands, the Sentinel-2 satellites [12] collect 12 bands, and the Hyperion satellite [35] collects 242 (hyperspectral) bands all measuring different regions of the electromagnetic spectrum. The exact wavelengths of the electromagnetic spectrum captured by each band can range from 400 nm to 15 μm. In addition, different sensors capture imagery at different spatial resolutions: satellite imagery can range from 4 km/px (GOES) to 30 cm/px (Maxar WorldView satellites), while imagery captured from drones can go as low as 7 mm/px. Depending on the type of orbit a satellite is in, imagery can be continuous (for geostationary orbits) or daily to biweekly (for polar, sun-synchronous orbits). Machine learning models or algorithms developed for one of these platforms will not generalize across inputs collected by the others, and, as a consequence of this, it is not possible to publish a single set of pre-trained model weights that span imaging platforms. In contrast, ImageNet [11] pretrained models have been proven to be useful in a large number of transfer learning tasks [57]. Researchers and practitioners can often start with ImageNet pre-trained models in a transfer learning setup when presented with vision problems to reduce the overall amount of effort needed to solve the problem. Further, it isn't clear whether the inductive biases built into common modeling approaches for vision problems are immediately applicable to remotely sensed imagery. Large neural architecture search efforts [25; 64] produce models that are optimized for, and indeed, outperform hand designed architectures on vision tasks, but it is an open question whether these transfer to remotely sensed imagery.

Most machine learning libraries have not been designed to work with geospatial data. For example, the Python Imaging Library (PIL) [8], used by many libraries to load images and perform data augmentation, does not support multispectral imagery. Similarly, deep learning models implemented by the torchvision library only support 3 channel (RGB) inputs, and must be adapted, or re-implemented to support multispectral data. Datasets of geospatial data can be made up of a heterogenous mix of files with differing file formats, spatial resolutions, projections, and coordinate reference systems (CRS). Libraries such as rasterio [17] and fiona [49] can interface with most types of geospatial data, however further abstractions for using such data in arbitrary deep learning pipelines are limited. Indeed, the gap between loading geospatial data from disk, and using it in a modeling pipeline, is large for all of the reasons mentioned above. For example, users will often need **pixel-aligned** crops from multiple layers of data: imagery from different points in time over the same space, imagery and corresponding label masks, high-resolution and low resolution imagery from the same space, etc. In contrast, there are a wide variety of software libraries at the intersection of machine learning and other domains. The PyTorch ecosystem alone has torchvision [37], torchtext [38], torchaudio [39], Hugging Face's Transformers [55], PyTorch Geometric [15], PyTorch

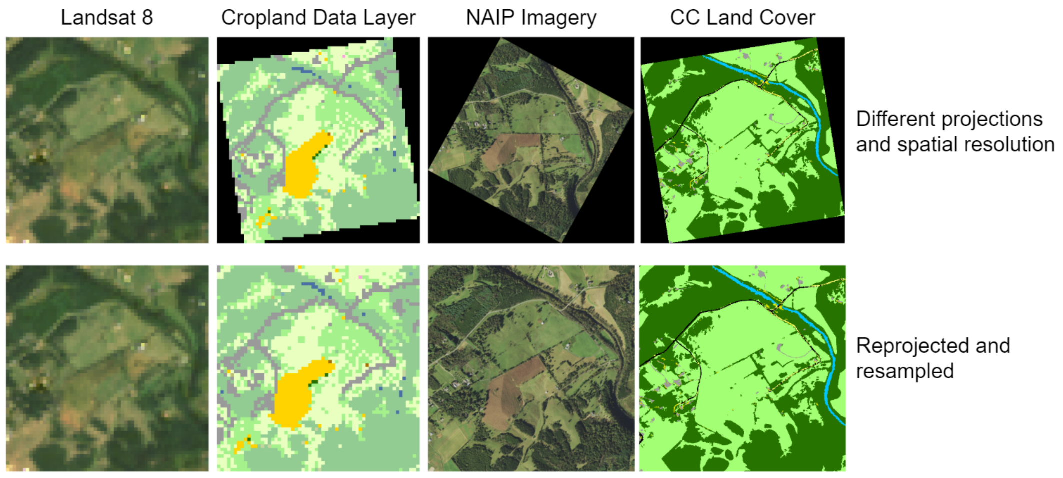
Landsat 8
Cropland Data Layer
NAIP Imagery
CC Land Cover
Different projections and spatial resolution
Reprojected and resampled

Figure 2: Different layers of geospatial data often have differing coordinate reference systems and spatial resolutions. (**Top row**) The *same physical area* cropped from four raster layers with different coordinate reference systems and spatial resolutions – this data is not pixel-aligned and cannot yet be used in modelling pipelines. (**Bottom row**) The same data as after reprojecting into the same coordinate system and resampling to the highest spatial resolution – this data is pixel aligned and can serve as inputs or masks to deep neural networks.

Meta [10], and PyTorch Video [14], that provide the tools necessary for abstracting the peculiarities of domain-specific data away from the details of deep learning training pipelines.

To address these challenges we propose TorchGeo, a Python package that allows users to transparently use heterogenous geospatial data in PyTorch-based deep learning pipelines. Specifically, TorchGeo provides:

1. data loaders for common geospatial datasets from the literature
2. *composable* data loaders for arbitrary geospatial raster and vector data with the ability to create **pixel-aligned** patches of data on-the-fly
3. augmentations that are appropriate for multispectral imagery
4. data samplers appropriate for geospatial data

We formally describe TorchGeo, propose and test methods for sampling from large geospatial datasets, and test the effect of ImageNet pretraining versus random weight initialization on several benchmark datasets. We achieve state of the art results on the the RESISC45 dataset, despite focusing only on creating simple and reproducible results to serve as baselines for future work to build on. We further find that ImageNet pre-training significantly improves spatial generalization performance in a land cover mapping task. We believe that these results are interesting in their own right, and that they highlight the importance of TorchGeo to the larger machine learning community.

2 Design

Remotely sensed imagery datasets are usually formatted as *scenes*, i.e. tensors $X \in \mathbb{R}^{H \times W \times C}$, where H is height, W is width, and C is the number of spectral channels, with corresponding spatial and temporal metadata. This metadata includes a *coordinate reference system* (CRS) that maps pixel coordinates to the surface of the Earth, a *spatial resolution* (the size of each pixel when mapped onto the surface of the earth), *spatial bounds* (a bounding box representing the area on Earth that the data covers), and a timestamp or time range to indicate when the data was collected. We say that two datasets, X^1 and X^2 are **pixel-aligned** if $X^1_{i,j}$ and $X^2_{i,j}$ represent data from the same positions on Earth for all i,j. Most pairs of datasets are **not** aligned by default. For example, X^1 and X^2 can be captured by two satellites in different orbits and will only have partially overlapping spatial bounds, or X^1 will be satellite imagery while X^2 will be from a dataset of labels with a different CRS (see Figure 2). However, deep learning model training requires **pixel-aligned** *patches* of imagery—i.e. smaller crops from large scenes. Most models are trained with mini-batch gradient descent and require input tensors in the format $B \times H \times W \times C$ where B is the number of samples in a mini-batch where W and H are constant over all samples in the batch. At a higher level, training semantic segmentation models will require pairs of pixel-aligned imagery and masks.

Aligning two datasets requires *reprojecting* the data from one in order to match the CRS of the other, *cropping* the data to the same spatial bounds, and *resampling* the data to correct for differences in resolution or to establish the same underlying pixel grid. Typically, these are performed as pre-processing steps using GIS software such as QGIS, ArcGIS, or tools provided by GDAL—see Section A.3 for an example of a GDAL command to align two data layers. This requires some level of domain knowledge to perform correctly and does not scale to large datasets as it requires creating duplicates of the layer to be aligned. Further, this approach still requires an implementation of a dataset or dataloader that can sample patches from the pre-processed imagery.

In TorchGeo, we facilitate this process by performing the alignment logic on-the-fly to create **pixel-aligned** *patches* of data sampled from larger scenes. Specifically, we implement the alignment logic in custom PyTorch **dataset** classes that are indexed in terms of spatial coordinates. Given a query patch in spatial coordinates, a desired destination CRS, and a desired spatial resolution, the custom dataset is responsible for returning the corresponding reprojected and resampled data for the query. We further implement geospatial data *samplers* that generate queries according to different criteria (e.g. randomly or in a regular grid pattern). See Section 3 for a discussion on the implementation of these.

As our datasets are indexed by spatial coordinates we can easily compose datasets that represent different data layers by specifying a valid area to sample from. For example, if we have two datasets, D^1 and D^2, we may want to sample data from the union of the layers, $D^1 \cup D^2$, if both layers are imagery, or the intersection of the layers, $D^1 \cap D^2$, if one layer is imagery and the other layer is labels. As we describe in the following section, we implement generic dataset classes for a variety of common remotely sensed datasets (e.g. Landsat imagery) that can be composed in this way. This allows users of the library to create their own multi-modal datasets without having to write custom code.

Most importantly, these abstractions that TorchGeo creates—geospatial datasets and samplers—can be combined with a standard PyTorch data loader class to produce fixed size batches of data to be transferred to the GPU and used in training or inference. See Listing 1 for a working example of TorchGeo code for setting up a functional data loader that uses Landsat and Cropland Data Layer (CDL) dataset implementations. Our approach trades off space for time compared to pre-processing all data layers and using a custom dataset described in the previous paragraph, but, crucially, does not require knowledge of GIS tooling. We benchmark our implementations in Section 4.3.

3 Implementation

The implementation of TorchGeo follows the design of other PyTorch domain libraries to reduce the amount of new concepts that a user must learn to integrate it with their existing workflow. We split TorchGeo up into the following parts:

Datasets Our dataset implementations consist of both *benchmark datasets* that allow users to interface with common datasets used in EO literature and *generic datasets* that allow users to interface with common geospatial data layers such as Landsat or Sentinel 2 imagery. Either of these types of datasets can also be *geospatial datasets*, i.e. datasets that contain geospatial metadata and can be sampled as such. These are part of the core contribution of TorchGeo and we describe them further in the following section.

Samplers We implement samplers for indexing into any of our *geospatial datasets*. Our geospatial datasets are indexed by bounding boxes in geospatial coordinates (as opposed to standard fixed length datasets of images which are usually indexed by an integer). The samplers generate bounding boxes according to specific patterns: randomly across all scenes in a dataset, random batches from single scenes at a time, or in grid patterns over scenes. Different sampling patterns can be useful for different model training strategies, or for running model inference over datasets.

Models Most existing model implementations (e.g. in torchvision) are fixed to accept 3 channel inputs which are not compatible with multispectral imagery. We provide implementations (or wrappers around well established implementations) of common deep learning model architectures with variable sized inputs and pre-trained weights. We also implement architectures from recent geospatial ML work such as the Fully Convolutional Siamese Network [9].

Transforms Similar to existing model implementations, some existing deep learning packages do not support data augmentation methods for multi-spectral imagery. We provide wrappers for augmentations in the Kornia [40] library (which does support augmentations over arbitrary channels). We also provide data transforms that generate spectral indices [20] from combinations of multispectral bands, for example: Normalized Difference Built-up Index (NDBI) [58], Normalized Difference Snow (NDSI) [41], Normalized Difference Vegetation (NDVI) [51], and Normalized Difference Water Index (NDWI) [30].

Trainers Finally, we implement model training recipes using the PyTorch Lightning library [54]. These include both dataset specific training code, and general training routines—for example, an implementation of the BYOL self-supervision method [18].

3.1 Datasets

We organize datasets based on whether they are a *generic dataset* or a *benchmark dataset* and based on whether or not they contain geospatial metadata—i.e. are a *geospatial dataset*.

Benchmark datasets are datasets released by the community that consist of both inputs and target labels for a specific type of task (scene classification, semantic segmentation, instance segmentation, etc.). These may or may not also contain geospatial metadata that allows them to be joined with other sources of data. In our opinion, one of the strongest components of existing deep learning domain libraries is the way that they make the use of existing datasets trivial. We aim to replicate this and, for example, include options that let users automatically download the data for a corresponding dataset. Table 1 lists the set of benchmark datasets that TorchGeo currently supports.

Dataset	Task	Source	# Samples	# Categories	Size (px)	Resolution (m)	Bands
ADVANCE [21]	C	Google Earth, Freesound	5,075	13	512x512	-	RGB
EuroSAT [19]	C	Sentinel-2	27,000	10	64x64	10	MSI
PatternNet [61]	C	Google Earth	30,400	38	256x256	0.06-5	RGB
RESISC45 [7]	C	Google Earth	31,500	45	256x256	0.2-30	RGB
So2Sat [63]	C	Sentinel-1, Sentinel-2	400,673	17	32x32	10	SAR & MSI
UC Merced [56]	C	USGS National Map	21,000	21	256x256	0.3	RGB
SEN12MS [46]	C	Sentinel-1, Sentinel-2, MODIS	180,662	33	256x256	10	SAR & MSI
CV4A Kenya Crop Type Competition [16]	SS	Sentinel-2	4,688	7	3,035x2,016	10	MSI
ETCI 2021 Competition on Flood Detection [33]	SS	Sentinel-1	66,810	2	256x256	5-20	SAR
GID-15 [50]	SS	Gaofen-2	150	15	6,800x7,200	3	RGB
LandCover.ai [2]	SS	Aerial	10,674	5	512x512	0.25-0.5	RGB
Smallholder Cashew Plantations in Benin [23]	SS	Airbus Pléiades	70	6	1186x1122	0.5	MSI
NWPU VHR-10 [6]	I	Google Earth, Vaihingen	800	10	358-1,728	0.08-2	RGB
SpaceNet 1 [13]	I	WorldView-2	6,940	2	102-439	0.5-1	MSI
SpaceNet 2 [13]	I	WorldView-3	10,592	2	162-650	0.3-1.24	MSI
ZueriCrop [52]	T, I	Sentinel-2	116,000	48	24x24	10	MSI
LEVIR-CD+ [47]	CD	Google Earth	985	2	1024x1024	0.5	RGB
COWC [32]	C, R	CSUAV AFRL, ISPRS, LINZ, AGRC	388,435	2	256x256	0.15	RGB
Tropical Cyclone Wind Estimation Competition [29]	R	GOES 8-16	108,110	-	256x256	4k-8k	MSI

C = classification, SS = semantic segmentation, I = instance segmentation, T = time series, CD = change detection, R = regression

Table 1: Benchmark datasets implemented in TorchGeo.

Generic datasets are not created with a specific task in mind, but instead represent layers of geospatial data that can be used for any purpose. For example, we implement datasets for representing collections of scenes of Landsat imagery that lets users index into the imagery and (combined with TorchGeo samplers) use it in arbitrary PyTorch based pipelines. These are not limited to imagery, for example we also implement a dataset representing the Cropland Data Layer labels, an annual US wide raster layer that gives the estimated crop type or land cover at a 30 m/px resolution.

Type	Dataset
Image Source	Landsat
	Sentinel
	National Agriculture Imagery Program (NAIP)
Labels	Cropland Data Layer (CDL)
	Chesapeake Land Cover
	Canadian Buildings Footprints

Table 2: Generic datasets implemented in TorchGeo.

3.2 SAMPLERS

As our *geospatial datasets* are indexed with bounding boxes using geospatial coordinates and do not have a concept of a dataset "length", they cannot be sampled from by choosing a random integer. We provide three types of samplers for different situations: a "RandomGeoSampler" that returns a fixed-sized bounding box from the valid spatial extent of a dataset uniformly at random, a "RandomBatchedGeoSampler" that returns a set of randomly positioned fixed-sized bounding boxes from a random scene within a dataset, and a "GridGeoSampler" that returns bounding-boxes in a grid pattern over subsequent scenes within a dataset. This abstraction also allows for methods that rely on specific data sampling patterns. For example, Tile2Vec [22] relies on sampling triplets of imagery where two of the images are close to each other in space while the third is distant. This logic can be implemented in several lines of code as a custom sampler class, that would then operate over any of the generic imagery datasets. Finally, all TorchGeo samplers are compatible with PyTorch dataloader objects so can be fit into any PyTorch based pipeline.

4 EXPERIMENTS AND RESULTS

4.1 DATASETS

We use the following datasets in our experiments:

Landsat and Cropland Data Layer (CDL) The Landsat and CDL dataset consists of multispectral imagery from 114 Landsat 8 [45] collection 2 level 2 scenes taken in 2019 that intersect with the continental United States and the 2019 Cropland Data Layer (CDL) dataset [4]. This data is 151 GB on disk and is stored in cloud optimized GeoTIFF (COG) format. We use this dataset to benchmark our proposed GeoDataset and sampler implementations.

So2Sat The So2Sat-LCZ42 dataset [62] is a *classification* dataset that consists of 400,673 image patches classified with one of 42 different local climate zone labels. The patches are 30×30 pixels in size with 18 channels consisting of Sentinel 1 and Sentinel 2 bands. They are sampled from different urban areas around the globe. We use the second version of the dataset as described on the project's GitHub page[1] in which the training split consists of data from 42 cities around the world, the validation split consists of the western half of 10 other cities, and the testing split covers the eastern half of the 10 remaining cities.

LandCover.AI The LandCover.AI dataset is a *semantic segmentation* dataset [3] that consists of high-resolution (0.5 m/px and 0.25 m/px) RGB aerial imagery from 41 *tiles* over Poland where each pixel has been classified as one of five classes (four land cover classes and a "background" class). The scenes are divided into 10,674 512×512 pixel patches and partitioned into pre-defined training, validation, and test splits according to the script on the dataset webpage[2].

Chesapeake Land Cover The Chesapeake Land Cover dataset [42] is a *semantic segmentation* dataset that consists of high-resolution (1 m/px) imagery from the US Department of Agriculture's National Agriculture Imagery Program (NAIP) and high-resolution (1 m/px) 6-class land cover labels from the Chesapeake Conservancy. Specifically, the dataset contains imagery and land

[1]`https://github.com/zhu-xlab/So2Sat-LCZ42`
[2]`https://landcover.ai`

cover masks for parts of six states in the Northeastern US: Maryland, Delaware, Virginia, West Virginia, Pensylvania, and New York. The data for each state is split into $\sim 7 \times 6$ km tiles and then divided into pre-defined training, validation, and test splits[3].

RESISC45 The RESISC45 dataset is a *classification* dataset [60] that consists of 31,500 256×256 pixel RGB image patches of varying spatial resolutions where each patch is classified into one of 45 classes. The dataset does not specify splits for training, validation or testing. In our experiments we generate 60/20/20 train/validation/test splits randomly.

4.2 Data loader benchmarks

We first benchmark the speed at which TorchGeo can sample patches of imagery and masks from the Landsat and CDL dataset. We believe this dataset is typical of a large class of geospatial machine learning problems—where users have access to a large amount of satellite imagery scenes covering a broad spatial extent and, separately, per pixel label masks where each scene is not necessarily projected in the same coordinate reference system. The end goal of such a problem is to train a model with *pixel-aligned* patches of imagery and label masks as described in Section 2. As such, we measure the rate that our dataset and sampler implementations can provide patches to a GPU for training and inference.

In Figure 3a, we calculate the rate at which samples of varying batch size can be drawn from a "GeoDataset" using various "GeoSampler" implementations. Compared to the other samplers, "GridGeoSampler" is significantly faster due to the repeated access of samples that are already in GDAL's cache. For small batch sizes, "RandomGeoSampler" and "RandomBatchGeoSampler" are almost identical, since overlap between patches is uncommon. However, for larger batch sizes, "RandomBatchGeoSampler" starts to outperform "RandomGeoSampler" as the cache is used more effectively.

In Figure 3b, we demonstrate the difference that preprocessing and caching data makes. This is most easily demonstrated by "GridGeoSampler" and to a lesser extent the other samplers. GDAL's cache only saves raw data loading times, so warping must always be done on-the-fly if the dataset CRSs or resolutions don't match. When the necessary storage is available, preprocessing the data ahead of time can lead to significantly faster sampling rates. Although "RandomGeoSampler" and "RandomBatchGeoSampler" are much slower than "GridGeoSampler", most users will only need to use "GridGeoSampler" for inference due to our pre-trained model weights.

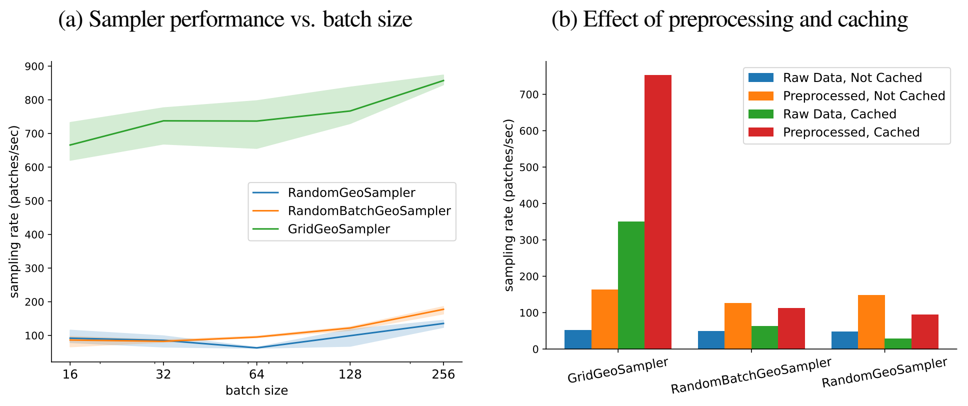

Figure 3: Sampling performance of various "GeoSampler" implementations. (a) Solid lines represent average sampling rate, while shaded region represents minimum and maximum performance across random seeds. (b) Average sampling rate under different data loading conditions.

4.3 Dataset benchmarks

We also use TorchGeo to create *reproducible benchmark results* on 4 of the datasets described in Section 4.1. There are many potential sources of variance in the results generated by a deep learning modeling

[3]`https://lila.science/datasets/chesapeakelandcover`

Dataset	Method	Pre-training	Bands	Performance
RESISC45 [7]	ResNet50	ImageNet	RGB	**98.20 +/- 0.16%**
	ResNet18	None	RGB	94.38 +/- 0.50%
	ResNet50 v2 [34]	In domain	RGB	96.86%
	ViT B/16 [48]	ImageNet-21k	RGB	96.80%
	ResNet50 [59]	Sup-Rotation-100%	RGB	96.30%
So2Sat [63]	ResNet50	ImageNet (+ random)	S2	**63.99 +/- 1.38%**
	ResNet50	None	S2	56.82 +/- 4.32%
	ResNet50	ImageNet	RGB	59.82 +/- 0.94%
	ResNet50	None	RGB	49.46 +/- 2.67%
	ResNeXt + CBAM [62]	None	S2	61%
	ResNet50 v2 [34]	In domain	RGB	**63.25%**
LandCover.AI [2]	U-Net, ResNet50 encoder	ImageNet	RGB	84.81 +/- 00.21%
	U-Net, ResNet50 encoder	None	RGB	79.73 +/- 00.67%
	DeepLabv3+, Xception71 with DPC encoder [2]	Cityscapes	RGB	**85.56%**
Chesapeake Land Cover [42] Delaware split	U-Net, ResNet50 encoder	ImageNet (+ random)	RGBN	**69.40 +/- 1.39%**
	U-Net, ResNet18 encoder	None	RGBN	**68.99 +/- 0.84%**

Table 3: Benchmark results comparing TorchGeo trained models to previously reported results over four datasets. RESISC and So2Sat results are reported as overall top-1 accuracy while LandCover.AI and Chesapeake Land Cover results are reported as mean class IoU. Results from TorchGeo models are reported as the mean with one standard deviation over 10 *training* runs from different random seeds. Results from related work are reported as is.

process, for example: the model architecture used, the type of optimizer used, the initial learning rate, training schedule, preprocessing methods, data augmentation methods, weight initialization, the pseudorandom number generator seed [36], and, importantly, implementation details of any of the previous points. If these sources of variance aren't controlled for then it can be impossible to know whether a proposed methodological innovation is the source of a difference (improvement) in performance on a particular benchmark dataset, or whether the difference is due to other factors in the model training pipeline. Our focus in these experiments is thus benchmarking simple models in a reproducible way and reporting the uncertainty in the results for future work to build on. To ensure reproducibility, we include a model training and evaluation framework in TorchGeo based around the PyTorch Lightning library and release all of our results. To quantify uncertainty we report the mean and standard deviation metrics calculated over 10 training runs with different random seeds.

For experiments, we use the pre-defined training, validation, and testing splits in the So2Sat, LandCover.AI, and Chesapeake Land Cover datasets, and use a random 60/20/20 split in the RESISC45 dataset. We perform a small hyperparameter search with a single fixed random seed on each dataset. Specifically we search over the best validation performance over the grid: learning rate $\in \{0.01, 0.001, 0.0001\}$, loss function $\in$ {cross entropy, jaccard}, weight initialization $\in$ {random, ImageNet} and model architecture $\in$ {ResNet18, ResNet50}[4]. With the So2Sat dataset we also run experiments that use all the Sentinel 2 bands vs. only the RGB bands. In the cases where we use ImageNet weights with imagery that has more than RGB bands we randomly initialize the non-RGB kernels in the first convolutional layer of the network and denote this setting as **ImageNet (+ random)**. In all cases we use the AdamW optimizer, reduce learning rate on validation loss plateaus, and early stop based on validation loss. We repeat the training process with 10 different seeds using the best performing hyperparameter configuration and report the test set performance over these models.

Our main results are shown in Table 3. We find that our simple training setup achieves state of the art results on RESISC45, although we note again that there is not a pre-defined train/val/test split for this dataset. Our splits were randomly drawn for each training seed and we report the average top-1 accuracy over 10 such seeds. The in-domain pre-training method in [34] trains models, starting with ImageNet weights, further on remote sensing (in domain) datasets before actually training on the target dataset and finds that this performs better than simply starting from ImageNet weights. In contrast, our best result comes from simply using Im-

[4]For the classification problems we use the ResNets *as is* and for the semantic segmentation datasets we use the ResNets as the encoder model in a U-Net.

ageNet pre-trained models and training on the target dataset with low learning rate. The difference between these two approaches is likely entirely due to different learning rate selection in the hyperparameter search and early stopping. On the So2Sat dataset we find the ImageNet pre-trained models are able to achieve similar results with in-domain pretraining, but only when using all S2 bands vs. RGB only. The previously reported baseline methods on the LandCover.AI dataset all use a DeepLabV3+ segmentation model with a Xception71 + Dense Prediction Cell (DPC) encoder that has been pre-trained on Cityscapes. We are able to achieve a result within 0.75 mIoU of this setup using a simple U-Net and ResNet50 encoder pre-trained on ImageNet. Finally, we report the first set of basic benchmark results on the Chesapeake Land Cover dataset.

4.4 Effect of ImageNet pre-training on generalization performance

Two of the datasets we test with, So2Sat and Chesapeake Land Cover, contain splits that are designed to measure the *generalization* performance of a model. The validation and test splits from So2Sat include data from urban areas that are *not* included in the training split while the Chesapeake Land Cover dataset contains separate splits for six different states. In these setting we observe a large performance boost when training models from ImageNet weights versus from a random initialization, however we *do not observe the boost* on in-domain data. Table 4 shows the performance of models that are trained on the Delaware split and evaluated on the test splits from every state. The in-domain performance (i.e. in Delaware) of the ImageNet pre-trained model and model trained from scratch are the same, however in every other setting the ImageNet pre-trained model performs better. In all cases but Maryland, this difference is greater than 6 points of mIoU with the most extreme difference being 12 points in Virginia. In the So2Sat case we find that most models are not able to significantly reduce validation loss (where in this case validation data is out-of-domain), while, unsurprisingly, achieving near perfect results on the training data. Despite this overfitting, there remains a large gap between the best and worst models, with ImageNet pre-trained models achieving +7% and +10% accuracy over randomly initialized models.

Weight init	**Delaware**	**Maryland**	**New York**	**Pennsylvania**	**Virginia**	**West Virginia**
ImageNet (+ random)	69.40 +/- 1.39%	59.57 +/- 0.70%	57.95 +/- 1.10%	55.13 +/- 1.25%	45.56 +/- 1.54%	20.76 +/- 1.95%
None	68.99 +/- 0.84%	57.30 +/- 0.78%	49.26 +/- 2.40%	47.67 +/- 2.40%	33.14 +/- 3.73%	14.95 +/- 2.72%

Table 4: Mean IoU performance of models trained in Delaware, with and without ImageNet weight initialization, on the test splits from Chesapeake Land Cover dataset.

5 Discussion

We introduce TorchGeo, a Python package for enabling deep learning with geospatial data. TorchGeo provides data loaders for common geospatial datasets, composable data loaders for arbitrary geospatial raster and vector data, samplers appropriate for geospatial data, models, transforms, and model trainers. Importantly, TorchGeo allows users to bypass the common pre-processing steps necessary to align geospatial data with imagery and performs this processing on-the-fly. We benchmark TorchGeo dataloader speed, and demonstrate how TorchGeo can be used to create reproducible benchmark results in several geospatial datasets.

Finally, TorchGeo serves as a platform for performing geospatial machine learning research. Existing work in self-supervised learning with geospatial data rely on spatiotemporal metadata and can be naturally implemented in TorchGeo and scaled over large amounts of geospatial imagery without the need for pre-processing steps. Similarly, augmentation methods appropriate for training with geospatial imagery are under-explored, however can be integrated easily with TorchGeo. Other interesting directions include building inductive biases appropriate for geospatial imagery into deep learning models (similar to the work done on rotation equivariant networks [28]), data fusion techniques (e.g. how to incorporate spatial information into models, or appropriately use multi-modal layers), and learning shape based models. Finally, TorchGeo exposes a catalog of benchmark geospatial datasets (Table 1) through a common interface, and, with the results in this paper, has begun to include corresponding benchmark results. This makes it easy for researchers to compare new ideas to existing work without having to repeat expensive computations. We hope TorchGeo can help drive advances at the intersection of machine learning and remote sensing.

REFERENCES

[1] Kumar Ayush, Burak Uzkent, Chenlin Meng, Kumar Tanmay, Marshall Burke, David Lobell, and Stefano Ermon. Geography-aware self-supervised learning. *arXiv preprint arXiv:2011.09980*, 2020.

[2] Adrian Boguszewski, Dominik Batorski, Natalia Ziemba-Jankowska, Anna Zambrzycka, and Tomasz Dziedzic. Landcover.ai: Dataset for automatic mapping of buildings, woodlands and water from aerial imagery. *CoRR*, abs/2005.02264, 2020. URL `https://arxiv.org/abs/2005.02264`.

[3] Adrian Boguszewski, Dominik Batorski, Natalia Ziemba-Jankowska, Tomasz Dziedzic, and Anna Zambrzycka. Landcover. ai: Dataset for automatic mapping of buildings, woodlands, water and roads from aerial imagery. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1102–1110, 2021.

[4] Claire Boryan, Zhengwei Yang, Rick Mueller, and Mike Craig. Monitoring us agriculture: the us department of agriculture, national agricultural statistics service, cropland data layer program. *Geocarto International*, 26(5):341–358, 2011.

[5] Gustau Camps-Valls, Devis Tuia, Xiao Xiang Zhu, and Markus Reichstein. *Deep learning for the Earth Sciences: A comprehensive approach to remote sensing, climate science and geosciences*. John Wiley & Sons, 2021.

[6] Gong Cheng, Junwei Han, Peicheng Zhou, and Lei Guo. Multi-class geospatial object detection and geographic image classification based on collection of part detectors. *ISPRS Journal of Photogrammetry and Remote Sensing*, 98:119–132, 2014. ISSN 0924-2716. doi: https://doi.org/10.1016/j.isprsjprs.2014.10.002. URL `https://www.sciencedirect.com/science/article/pii/S0924271614002524`.

[7] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *Proceedings of the IEEE*, 105(10):1865–1883, 2017. doi: 10.1109/JPROC.2017.2675998.

[8] Alex Clark. Pillow. `https://github.com/python-pillow/Pillow`, 2010.

[9] Rodrigo Caye Daudt, Bertrand Le Saux, and Alexandre Boulch. Fully convolutional siamese networks for change detection. *CoRR*, abs/1810.08462, 2018. URL `http://arxiv.org/abs/1810.08462`.

[10] Tristan Deleu, Tobias Würfl, Mandana Samiei, Joseph Paul Cohen, and Yoshua Bengio. Torchmeta: A meta-learning library for pytorch. *arXiv preprint arXiv:1909.06576*, 2019.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

[12] Matthias Drusch, Umberto Del Bello, Sébastien Carlier, Olivier Colin, Veronica Fernandez, Ferran Gascon, Bianca Hoersch, Claudia Isola, Paolo Laberinti, Philippe Martimort, et al. Sentinel-2: Esa's optical high-resolution mission for gmes operational services. *Remote sensing of Environment*, 120: 25–36, 2012.

[13] Adam Van Etten, Dave Lindenbaum, and Todd M. Bacastow. Spacenet: A remote sensing dataset and challenge series. *CoRR*, abs/1807.01232, 2018. URL `http://arxiv.org/abs/1807.01232`.

[14] Haoqi Fan, Tullie Murrell, Heng Wang, Kalyan Vasudev Alwala, Yanghao Li, Yilei Li, Bo Xiong, Nikhila Ravi, Meng Li, Haichuan Yang, Jitendra Malik, Ross Girshick, Matt Feiszli, Aaron Adcock, Wan-Yen Lo, and Christoph Feichtenhofer. PyTorchVideo: A deep learning library for video understanding. In *Proceedings of the 29th ACM International Conference on Multimedia*, 2021. `https://pytorchvideo.org/`.

[15] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[16] Radiant Earth Foundation. CV4A competition kenya crop type dataset, 2020. Version 1.0, Radiant MLHub.

[17] Sean Gillies, B Ward, AS Petersen, et al. Rasterio: Geospatial raster i/o for python programmers. *Mapbox.*, 2013.

[18] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

[19] Patrick Helber, Benjamin Bischke, Andreas Dengel, and Damian 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*, 12(7):2217–2226, 2019.

[20] V Henrich, E Götze, A Jung, C Sandow, D Thürkow, and C Gläßer. Development of an online indices database: Motivation, concept and implementation. In *Proceedings of the 6th EARSeL Imaging Spectroscopy SIG Workshop Innovative Tool for Scientific and Commercial Environment Applications, Tel Aviv, Israel*, pp. 16–18, 2009.

[21] Di Hu, Xuhong Li, Lichao Mou, Pu Jin, Dong Chen, Liping Jing, Xiaoxiang Zhu, and Dejing Dou. Cross-task transfer for geotagged audiovisual aerial scene recognition. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *Computer Vision – ECCV 2020*, pp. 68–84, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58586-0.

[22] Neal Jean, Sherrie Wang, Anshul Samar, George Azzari, David Lobell, and Stefano Ermon. Tile2vec: Unsupervised representation learning for spatially distributed data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3967–3974, 2019.

[23] Z. Jin, C. Lin, C. Weigl, J. Obarowski, and D. Hale. Smallholder cashew plantations in benin, 2021. Radiant MLHub.

[24] K Karra, C Kontgis, Z Statman-Weil, J Mazzariello, M Mathis, and S Brumby. Global land use/land cover with sentinel-2 and deep learning. In *Proceedings of the IGARSS*, volume 2021, 2021.

[25] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 19–34, 2018.

[26] Dengsheng Lu, Qi Chen, Guangxing Wang, Lijuan Liu, Guiying Li, and Emilio Moran. A survey of remote sensing-based aboveground biomass estimation methods in forest ecosystems. *International Journal of Digital Earth*, 9(1):63–105, 2016.

[27] Oscar Mañas, Alexandre Lacoste, Xavier Giro-i Nieto, David Vazquez, and Pau Rodriguez. Seasonal contrast: Unsupervised pre-training from uncurated remote sensing data. *arXiv preprint arXiv:2103.16607*, 2021.

[28] Diego Marcos, Michele Volpi, Benjamin Kellenberger, and Devis Tuia. Land cover mapping at very high resolution with rotation equivariant cnns: Towards small yet accurate models. *ISPRS journal of photogrammetry and remote sensing*, 145:96–107, 2018.

[29] Manil Maskey, Rahul Ramachandran, Muthukumaran Ramasubramanian, Iksha Gurung, Brian Freitag, Aaron Kaulfus, Drew Bollinger, Daniel J. Cecil, and Jeffrey Miller. Deepti: Deep-learning-based tropical cyclone intensity estimation system. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:4271–4281, 2020. doi: 10.1109/JSTARS.2020.3011907.

[30] Stuart K McFeeters. The use of the normalized difference water index (ndwi) in the delineation of open water features. *International journal of remote sensing*, 17(7):1425–1432, 1996.

[31] David J. Mulla. Twenty five years of remote sensing in precision agriculture: Key advances and remaining knowledge gaps. *Biosystems Engineering*, 114(4):358–371, 2013. ISSN 1537-5110. doi: https://doi.org/10.1016/j.biosystemseng.2012.08.009. URL `https://www.sciencedirect.com/science/article/pii/S1537511012001419`. Special Issue: Sensing Technologies for Sustainable Agriculture.

[32] T. Nathan Mundhenk, Goran Konjevod, Wesam A. Sakla, and Kofi Boakye. A large contextual dataset for classification, detection and counting of cars with deep learning. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *Computer Vision – ECCV 2016*, pp. 785–800, Cham, 2016. Springer International Publishing. ISBN 978-3-319-46487-9.

[33] NASA-IMPACT. Etci 2021 competition on flood detection. `https://nasa-impact.github.io/etci2021/`, 2021.

[34] Maxim Neumann, Andre Susano Pinto, Xiaohua Zhai, and Neil Houlsby. In-domain representation learning for remote sensing. *arXiv preprint arXiv:1911.06721*, 2019.

[35] Jay S Pearlman, Pamela S Barry, Carol C Segal, John Shepanski, Debra Beiso, and Stephen L Carman. Hyperion, a space-based imaging spectrometer. *IEEE Transactions on Geoscience and Remote Sensing*, 41(6):1160–1173, 2003.

[36] David Picard. Torch. manual_seed (3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*, 2021.

[37] PyTorch. torchvision. `https://github.com/pytorch/vision`, 2016.

[38] PyTorch. torchtext. `https://github.com/pytorch/text`, 2017.

[39] PyTorch. torchaudio. `https://github.com/pytorch/audio`, 2019.

[40] Edgar Riba, Dmytro Mishkin, Daniel Ponsa, Ethan Rublee, and Gary Bradski. Kornia: an open source differentiable computer vision library for pytorch. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 3674–3683, 2020.

[41] George A Riggs, Dorothy K Hall, and Vincent V Salomonson. A snow index for the landsat thematic mapper and moderate resolution imaging spectroradiometer. In *Proceedings of IGARSS'94-1994 IEEE International Geoscience and Remote Sensing Symposium*, volume 4, pp. 1942–1944. IEEE, 1994.

[42] Caleb Robinson, Le Hou, Kolya Malkin, Rachel Soobitsky, Jacob Czawlytko, Bistra Dilkina, and Nebojsa Jojic. Large scale high-resolution land cover mapping with multi-resolution data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 12726–12735, 2019.

[43] John Rogan, Janet Franklin, Doug Stow, Jennifer Miller, Curtis Woodcock, and Dar Roberts. Mapping land-cover modifications over large areas: A comparison of machine learning algorithms. *Remote Sensing of Environment*, 112(5):2272–2283, 2008.

[44] David Rolnick, Priya L. Donti, Lynn H. Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, Alexandra Luccioni, Tegan Maharaj, Evan D. Sherwin, S. Karthik Mukkavilli, Konrad P. Körding, Carla P. Gomes, Andrew Y. Ng, Demis Hassabis, John C. Platt, Felix Creutzig, Jennifer T. Chayes, and Yoshua Bengio. Tackling climate change with machine learning. *CoRR*, abs/1906.05433, 2019. URL `http://arxiv.org/abs/1906.05433`.

[45] David P Roy, Michael A Wulder, Thomas R Loveland, Curtis E Woodcock, Richard G Allen, Martha C Anderson, Dennis Helder, James R Irons, David M Johnson, Robert Kennedy, et al. Landsat-8: Science and product vision for terrestrial global change research. *Remote sensing of Environment*, 145:154–172, 2014.

[46] M. Schmitt, L. H. Hughes, C. Qiu, and X. X. Zhu. Sen12ms – a curated dataset of georeferenced multi-spectral sentinel-1/2 imagery for deep learning and data fusion. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W7:153–160, 2019. doi: 10.5194/isprs-annals-IV-2-W7-153-2019. URL `https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-2-W7/153/2019/`.

[47] Li Shen, Yao Lu, Hao Chen, Hao Wei, Donghai Xie, Jiabao Yue, Rui Chen, Yue Zhang, Ao Zhang, Shouye Lv, and Bitao Jiang. S2looking: A satellite side-looking dataset for building change detection. *CoRR*, abs/2107.09244, 2021. URL `https://arxiv.org/abs/2107.09244`.

[48] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.

[49] Toblerity. fiona. `https://github.com/Toblerity/Fiona`, 2011.

[50] Xin-Yi Tong, Gui-Song Xia, Qikai Lu, Huanfeng Shen, Shengyang Li, Shucheng You, and Liangpei Zhang. Land-cover classification with high-resolution remote sensing images using transferable deep models. *Remote Sensing of Environment*, 237:111322, 2020. ISSN 0034-4257. doi: https://doi.org/10.1016/j.rse.2019.111322. URL `https://www.sciencedirect.com/science/article/pii/S0034425719303414`.

[51] Compton J Tucker. Red and photographic infrared linear combinations for monitoring vegetation. *Remote sensing of Environment*, 8(2):127–150, 1979.

[52] Mehmet Ozgur Turkoglu, Stefano D'Aronco, Gregor Perich, Frank Liebisch, Constantin Streit, Konrad Schindler, and Jan Dirk Wegner. Crop mapping from image time series: Deep learning with multi-scale label hierarchies. *Remote Sensing of Environment*, 264:112603, 2021. ISSN 0034-4257. doi: https://doi.org/10.1016/j.rse.2021.112603. URL `https://www.sciencedirect.com/science/article/pii/S0034425721003230`.

[53] Cees J Van Westen. Remote sensing and gis for natural hazards assessment and disaster risk management. *Treatise on geomorphology*, 3:259–298, 2013.

[54] Falcon William and The PyTorch Lightning team. PyTorch Lightning, 3 2019. URL `https://github.com/PyTorchLightning/pytorch-lightning`.

[55] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[56] Yi Yang and Shawn Newsam. Bag-of-visual-words and spatial extensions for land-use classification. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, pp. 270–279, 2010.

[57] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *arXiv preprint arXiv:1411.1792*, 2014.

[58] Yong Zha, Jay Gao, and Shaoxiang Ni. Use of normalized difference built-up index in automatically mapping urban areas from tm imagery. *International journal of remote sensing*, 24(3):583–594, 2003.

[59] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[60] Wei Zhang, Ping Tang, and Lijun Zhao. Remote sensing image scene classification using cnn-capsnet. *Remote Sensing*, 11(5):494, 2019.

[61] Weixun Zhou, Shawn Newsam, Congmin Li, and Zhenfeng Shao. Patternnet: A benchmark dataset for performance evaluation of remote sensing image retrieval. *ISPRS Journal of Photogrammetry and Remote Sensing*, 145:197–209, 2018. ISSN 0924-2716. doi: https://doi.org/10.1016/j.isprsjprs.2018.01.004. URL `https://www.sciencedirect.com/science/article/pii/S0924271618300042`. Deep Learning RS Data.

[62] Xiao Xiang Zhu, Jingliang Hu, Chunping Qiu, Yilei Shi, Jian Kang, Lichao Mou, Hossein Bagheri, Matthias Häberle, Yuansheng Hua, Rong Huang, et al. So2sat lcz42: A benchmark dataset for global local climate zones classification. *arXiv preprint arXiv:1912.12171*, 2019.

[63] Xiao Xiang Zhu, Jingliang Hu, Chunping Qiu, Yilei Shi, Jian Kang, Lichao Mou, Hossein Bagheri, Matthias Haberle, Yuansheng Hua, Rong Huang, Lloyd Hughes, Hao Li, Yao Sun, Guichen Zhang, Shiyao Han, Michael Schmitt, and Yuanyuan Wang. So2sat lcz42: A benchmark data set for the classification of global local climate zones [software and data sets]. *IEEE Geoscience and Remote Sensing Magazine*, 8(3):76–89, 2020. doi: 10.1109/MGRS.2020.2964708.

[64] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

A Appendix

A.1 Data loader benchmarking

In order to benchmark the performance of our data loaders, we download 114 Landsat 8 scenes and 1 Cropland Data Layer (CDL) file for the year of 2019. All files are stored as Cloud Optimized GeoTIFFs (COGs) and kept in their original CRS (Albers Equal Area for CDL and UTM for Landsat). Experiments are run on Microsoft Azure with a 6-core Intel Xeon E5-2690 CPU. All data is stored on a local SSD attached to the compute node. Batch size and random seed are varied while the remaining hyperparameters are kept fixed. Total epoch size is 4096, patch size is 224, stride is 112, and number of workers for parallel data loading is set to 6.

A.2 TorchGeo code example

```python
from torch.utils.data import DataLoader
from torchgeo.datasets import Landsat8, CDL
from torchgeo.samplers import RandomGeoSampler

# Instantiate CDL and Landsat 8 Datasets
cdl = CDL(root="...")
landsat = Landsat8(root="...", crs=cdl.crs, res=cdl.res)
merged = cdl + landsat  # spatial merge

# Sample 1 km^2 image chips from the intersection of the datasets
sampler = RandomGeoSampler(merged, size=1000, length=512)

# Use the dataset and sampler as normal in a PyTorch DataLoader
dataloader = DataLoader(merged, sampler=sampler, batch_size=32)
```

Listing 1: Example TorchGeo code for creating a joint Landsat 8 and Cropland Data Layer (CDL) dataset, and using such a dataset with a standard PyTorch DataLoader class.

A.3 Pre-processing alignment with GDAL

As an example of an alignment pre-processing workflow, we assume that we have a Landsat 8 scene and a Cropland Data Layer (CDL) raster ("cdl.tif") which completely covers the extent of the Landsat scene. We would like to create a **pixel-aligned** version of these two layers. Given that the Landsat 8 scene has a CRS of "EPSG:32619", a height of 8011 pixels, a width of 7891 pixels, and spatial bounds of (186585,4505085,423315,4745415), the corresponding GDAL command would create a cropped version of the CDL layer that is aligned to the Landsat layer:

```bash
gdalwarp \
    -t_srs EPSG:32619 \
    -of COG \
    -te 186585 4505085 423315 4745415 \
    -ts 7891 8011 \
    cdl.tif aligned_cdl.tif
```

The spatial metadata of the Landsat scene can be determined through other GDAL command line tools (gdalinfo command), geospatial data packages such as the "rasterio" package in Python, or through GIS software such as QGIS or ArcGIS.