AiTLAS

Artificial Intelligence Toolbox for Earth Observation

Documentation

Bias Variance Labs
www.bvlabs.ai

July, 2023

Contents

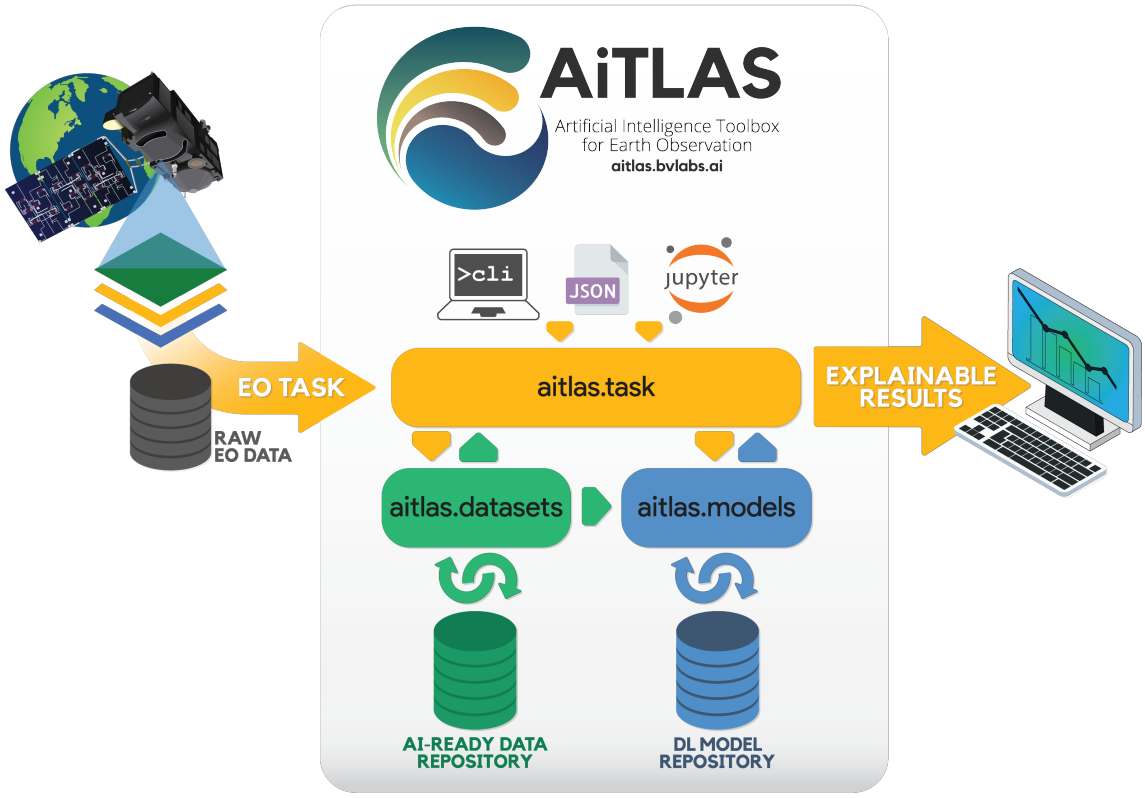
Contents

CHAPTER 1

AiTLAS: Artificial Intelligence Toolbox for Earth Observation

The AiTLAS toolbox (Artificial Intelligence Toolbox for Earth Observation) includes state-of-the-art machine learning methods for exploratory and predictive analysis of satellite imagery as well as repository of AI-ready Earth Observation (EO) datasets. It can be easily applied for a variety of Earth Observation tasks, such as land use and cover classification, crop type prediction, localization of specific objects (semantic segmentation), etc. The main goal of AiTLAS is to facilitate better usability and adoption of novel AI methods (and models) by EO experts, while offering easy access and standardized format of EO datasets to AI experts which allows benchmarking of various existing and novel AI methods tailored for EO data.

AiTLAS adheres to the principle *less is more* - it embeds the most common tasks and functionalities in easy-to-use interfaces that simplify the usage and adaptation of the toolbox with minimal modifications. It is fully aligned with the principles of open-science and open-software.



1.1 Getting started

- AiTLAS Introduction[1]
- AiTLAS Software Architecture[2]
- AiTLAS in a nutshell[3]

AiTLAS examples:

- Land use classification with multi-class classification with AiTLAS[4]
- Land use classification with multi-label classification[5]
- Maya archeological sites segmentation[6]
- Visualization of learning performance[7]

[1] https://youtu.be/-3Son1NhdDg
[2] https://youtu.be/cLfEZFQQiXc
[3] https://www.youtube.com/watch?v=lhDjiZg7RwU
[4] https://youtu.be/JcJXrMch0Rc
[5] https://youtu.be/yzHkEMbDW7s
[6] https://youtu.be/LBFY4pCfzOU
[7] https://youtu.be/wjMfstcWBSs

1.2 Contact

For any questions and issues feel free to contact us via email at info@bvlabs.ai or by opening an issue on the official repository[8].

1.3 Licence

Apache-2.0 license[9]

1.4 Citation

For attribution in academic contexts, please cite our work **'AiTLAS: Artificial Intelligence Toolbox for Earth Observation'** published in Remote Sensing (2023) link[10] as

```
@article{aitlas2023,
AUTHOR = {Dimitrovski, Ivica and Kitanovski, Ivan and Panov, Panče and
↪Kostovska, Ana and Simidjievski, Nikola and Kocev, Dragi},
TITLE = {AiTLAS: Artificial Intelligence Toolbox for Earth Observation},
JOURNAL = {Remote Sensing},
VOLUME = {15},
YEAR = {2023},
NUMBER = {9},
ARTICLE-NUMBER = {2343},
ISSN = {2072-4292},
DOI = {10.3390/rs15092343}
}
```

1.5 Acknowlidgments

The AiTLAS toolbox is developed within the grant from the European Space Agency (ESRIN): AiTLAS–Artificial Intelligence toolbox for Earth Observation (ESA RFP/3-16371/19/I-NB) awarded to Bias Variance Labs, d.o.o[11].

[8] https://github.com/biasvariancelabs/aitlas
[9] https://github.com/biasvariancelabs/aitlas/blob/master/LICENSE
[10] https://www.mdpi.com/2072-4292/15/9/2343
[11] https://bvlabs.ai/

CHAPTER 2

What is AiTLAS?

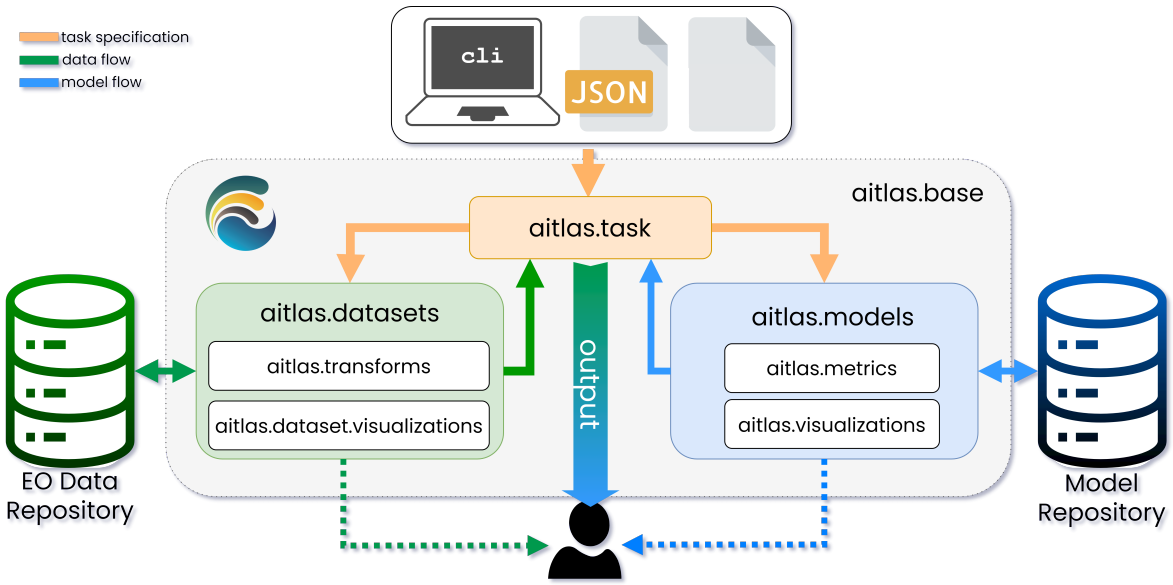
The AiTLAS toolbox (Artificial Intelligence Toolbox for Earth Observation) includes state-of-the-art machine learning methods for exploratory and predictive analysis of satellite imagery as well as repository of AI-ready Earth Observation (EO) datasets. It can be easily applied for a variety of Earth Observation tasks, such as land use and cover classification, crop type prediction, localization of specific objects (semantic segmentation), etc.

The main goal of AiTLAS is to facilitate better usability and adoption of novel AI methods (and models) by EO experts, while offering easy access and standardized format of EO datasets to AI experts which allows benchmarking of various existing and novel AI methods tailored for EO data.

2.1 Design

AiTLAS is designed such that leveraging recent (and sophisticated) deep-learning approaches over a variety of EO tasks (and data) is straightforward. On the one hand, it utilizes EO data resources in an AI-ready form; on the other hand, it provides a sufficient layer of abstraction for building and executing data analysis pipelines, thus facilitating better usability and accessibility of the underlying approaches - particularly useful for users with limited experience in machine learning, and in particular deep learning.

It can be used both as an end-to-end standalone tool and as a modular library. Users can use and build on different toolbox components independently, be they related to the tasks, datasets, models, benchmarks, or complete pipelines. It is also flexible and versatile, facilitating the execution of a wide array of tasks on various domains and providing easy extension and adaptation to novel tasks and domains.



AiTLAS is designed around the concept of a workflow, where users need to define a specific `task` be it an exploratory analysis of a dataset or a predictive task of a different kind, such as image classification, object detection, image segmentation, etc. The official repository[12], includes many different configuration files[13] for running various different workflows.

The instantiated task serves as an arbiter of the workflow and orchestrates the flow between the two central components of the toolbox - the datasets (`aitlas.datasets`) and the models (`aitlas.models`) - which relate to AI-ready formalized data and configurable model architectures, respectively. Programmatically, these modules are embedded within the core module `aitlas.base`, which contains all main abstract definitions related to every module, such as definitions of tasks, models, and datasets, but are also related to evaluations (`aitlas.metrics`), data transformations (`aitlas.transforms`), and various types of visualizations (`aitlas.visulizataions` and `aitlas.datasets.visulizataions`).

[12] https://github.com/biasvariancelabs/aitlas
[13] https://github.com/biasvariancelabs/aitlas/tree/master/configs

CHAPTER 3

The AiTLAS Ecosystem

3.1 AiTLAS: Benchmark Arena

An open-source benchmark framework for evaluating state-of-the-art deep learning approaches for image classification in Earth Observation (EO). To this end, it presents a comprehensive comparative analysis of more than 500 models derived from ten different state-of-the-art architectures and compare them to a variety of multi-class and multi-label classification tasks from 22 datasets with different sizes and properties. In addition to models trained entirely on these datasets, it employs benchmark models trained in the context of transfer learning, leveraging pre-trained model variants, as it is typically performed in practice. All presented approaches are general and can be easily extended to many other remote sensing image classification tasks.To ensure reproducibility and facilitate better usability and further developments, all of the experimental resources including the trained models, model configurations and processing details of the datasets (with their corresponding splits used for training and evaluating the models) are available on this repository.

repository: https://github.com/biasvariancelabs/aitlas-arena

paper: Current Trends in Deep Learning for Earth Observation: An Open-source Benchmark Arena for Image Classification[14] , ISPRS Journal of Photogrammetry and Remote Sensing, Vol.197, pp 18-35

3.2 AiTLAS Semantic Data Catalog of Earth Observation (EO) datasets (beta)

A novel semantic data catalog of numerous EO datasets, pertaining to various different EO and ML tasks. The catalog, that includes properties of different datasets and provides further details for their use, is available at eodata.bvlabs.ai[15]

[14] https://www.sciencedirect.com/science/article/pii/S0924271623000205
[15] http://eodata.bvlabs.ai

CHAPTER 4

AiTLAS installation

The best way to install `aitlas`, is if you create a virtual environment and install the requirements with `pip`. Here are the steps:

- Go to the folder where you cloned the repo.
- Create a virtual environment

```
conda create -n aitlas python=3.8
```

- Use the virtual environment

```
conda activate aitlas
```

- Before installing `aitlas` on Windows it is recommended to install the following packages from Unofficial Windows wheels repository[16]:

```
pip install GDAL-3.4.1-cp38-cp38-win_amd64.whl
pip install Fiona-1.8.20-cp38-cp38-win_amd64.whl
pip install rasterio-1.2.10-cp38-cp38-win_amd64.whl
```

- Install the requirements

```
pip install -r requirements.txt
```

And, that's it, you can start using `aitlas`!

```
python -m aitlas.run configs/example_config.json
```

If you want to use `aitlas` as a package run

```
pip install .
```

in the folder where you cloned the repo.

[16] https://www.lfd.uci.edu/~gohlke/pythonlibs/

CHAPTER 5

API documentation

5.1 Base module

5.1.1 Base module

aitlas.base.classification module

class aitlas.base.classification.**BaseMulticlassClassifier**(*config*)

Bases: *BaseModel*

Base class for a multiclass classifier.

BaseModel constructor

Parameters
config (*Config, optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *BaseClassifierSchema*

get_predicted(*outputs*, *threshold=None*)

Get predicted classes from the model outputs.

Parameters

- **outputs** (*torch.Tensor*) – Model outputs with shape (batch_size, num_classes).
- **threshold** (*float, optional*) – The threshold for classification, defaults to None.

Returns
tuple containing the probabilities and predicted classes

Return type
tuple

report(*labels*, *dataset_name*, *running_metrics*, ***kwargs*)

Generate a report for multiclass classification.

Parameters

- **labels** (*list*) – List of class labels.
- **dataset_name** (*list*) – Name of the dataset.
- **running_metrics** (aitlas.base.metrics.RunningScore) – A running score object for multiclass classification.

load_optimizer()

Load the optimizer

load_criterion()

Load the loss function

load_lr_scheduler(*optimizer*)

Load the learning rate scheduler

training: bool

class aitlas.base.classification.**BaseMultilabelClassifier**(*config*)

Bases: *BaseModel*

Base class for a multilabel classifier.

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *BaseClassifierSchema*

load_optimizer()

Load the optimizer

load_criterion()

Load the loss function

load_lr_scheduler(*optimizer*)

get_predicted(*outputs*, *threshold=None*)

Get predicted classes from the model outputs.

Parameters

- **outputs** (*torch.Tensor*) – Model outputs with shape (batch_size, num_classes).
- **threshold** (*float, optional*) – Threshold for classification, defaults to None

Returns

Tuple containing the probabilities and predicted classes.

Return type

tuple

report(*labels*, *dataset_name*, *running_metrics*, ***kwargs*)

Generate a report for multilabel classification.

Parameters

- **labels** (*list*) – List of class labels

- **dataset_name** (*str*) – Name of the dataset.
- **running_metrics** (aitlas.base.metrics.RunningScore) – Type of metrics to be reported. Currently only confusion matrix is

training: bool

aitlas.base.config module

class aitlas.base.config.**Config**(*config*)

Bases: Munch

Config object used for automatic object creation from a dictionary.

class aitlas.base.config.**ObjectConfig**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.base.config.**RunConfig**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Top level configuration schema

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.base.config.**Configurable**(*config*)

Bases: ABC

Base class for all configurable objects.

schema = None

aitlas.base.datasets module

Dataset base class.

This is the base class for all datasets. All datasets should subclass it.

class aitlas.base.datasets.**BaseDataset**(*config*)

Bases: Dataset, *Configurable*

This class represents a basic dataset for machine learning tasks. It is a subclass of both :class:Dataset and :class:Configurable. You can use it as a base class to define your own custom datasets.

Parameters

- **Dataset** (*_type_*) – _description_
- **Configurable** (*_type_*) – _description_

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *BaseDatasetSchema*

name = None

labels = None

get_name()

prepare()

Implement if something needs to happen to the dataset after object creation

dataloader()

Create and return a dataloader for the dataset

get_labels()

Implement this if you want to return the complete set of labels of the dataset

show_batch(*size*)

Implement this if you want to return a random batch of images from the dataset

show_samples()

Implement this if you want to return a random samples from the dataset

show_image(*index*)

Implement this if you want to return an image with a given index from the dataset

data_distribution_table()

Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()
Implement this if you want to return the label distribution of the dataset as a barchart

load_transforms(*class_names*)
Loads transformation classes and make a composition of them

aitlas.base.metrics module

class aitlas.base.metrics.**BaseMetric**(*device='cpu', **kwargs*)
Bases: object

Base class for metrics

calculate(*y_true, y_pred*)

class aitlas.base.metrics.**RunningScore**(*num_classes, device*)
Bases: object

update(*y_true, y_pred, y_prob=None*)
Updates stats on each batch

reset()
Reset the confusion matrix

get_computed()

precision()

accuracy()

weights()

recall()

f1_score()

iou()

get_scores(*metrics*)
Returns the specified metrics

class aitlas.base.metrics.**MultiClassRunningScore**(*num_classes, device*)
Bases: *RunningScore*

Calculates confusion matrix for multi-class data. This class contains metrics that are averaged over batches.

accuracy()

weights()

recall()

precision()

iou()

kappa()

class aitlas.base.metrics.**MultiLabelRunningScore**(*num_classes, device*)
Bases: *RunningScore*

Calculates a confusion matrix for multi-labelled, multi-class data in addition to the

reset()
Reset the confusion matrix and list of probabilities

update(*y_true*, *y_pred*, *y_prob=None*)
Updates stats on each batch

map()

roc_auc_score()

accuracy()

precision()

weights()

recall()

get_outcomes(*total=False*)
Return true/false positives/negatives from the confusion matrix :param total: do we need to return per class or total

count()

get_samples()

iou()

class aitlas.base.metrics.**SegmentationRunningScore**(*num_classes*, *device*)
Bases: *MultiLabelRunningScore*

Calculates a metrics for semantic segmentation

update(*y_true*, *y_pred*, *y_prob=None*)
Updates stats on each batch

class aitlas.base.metrics.**ObjectDetectionRunningScore**(*num_classes*, *device*)
Bases: object

Calculates a metrics for object detection

update(*preds*, *target*)
Updates stats on each batch

reset()
Reset the confusion matrix

compute()

map()
Returns the specified metrics

map_50()
Returns the specified metrics

get_scores(*metrics*)
Returns the specified metrics

aitlas.base.models module

Models base class. This is the base class for all models. All models should subclass it.

class aitlas.base.models.**EarlyStopping**(*patience=10*, *min_delta=0*)

Bases: object

Early stopping to stop the training when the loss does not improve after certain epochs.

BaseModel constructor

Parameters

- **patience** – how many epochs to wait before stopping when loss is not improving
- **min_delta** – minimum difference between new loss and old loss for new loss to be considered as an improvement

class aitlas.base.models.**BaseModel**(*config=None*)

Bases: Module, *Configurable*

Basic class abstracting a model. Contains methods for training, evaluation and also utility methods for loading, saving a model to storage.

BaseModel constructor

Parameters

config (*Config, optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *BaseModelSchema*

name = None

log_loss = True

prepare()

Prepare the model before using it. Loans loss criteria, optimizer, lr scheduler and early stopping.

fit(*dataset*, *epochs=100*, *model_directory=None*, *save_epochs=10*, *iterations_log=100*, *resume_model=None*, *val_dataset=None*, *run_id=None*, ***kwargs*)

Main method to train the model. It trains the model for the specified number of epochs and saves the model after every save_epochs. It also logs the loss after every iterations_log.

Parameters

- **dataset** (*aitlas.base.BaseDataset*) – Dataset object which contains the training data.
- **epochs** (*int, optional*) – Number of epochs to train the model, defaults to 100
- **model_directory** (*str, optional*) – Location where the model checkpoints will be stored or should be loaded from, defaults to None
- **save_epochs** (*int, optional*) – Number of epoch after a checkpoint is saved, defaults to 10
- **iterations_log** (*int, optional*) – Number of iteration after which the training status will be logged, defaults to 100
- **resume_model** (*str, optional*) – Whether or not to resume training a saved model, defaults to None

- **val_dataset** (*aitlas.base.BaseDataset, optional*) – Dataset object which contains the validation data., defaults to None
- **run_id** (*str, optional*) – Optional id to idenfity the experiment, defaults to None

Returns
Returns the loss at the end of training.

Return type
float

train_epoch(*epoch*, *dataloader*, *optimizer*, *criterion*, *iterations_log*)

evaluate(*dataset=None*, *model_path=None*)

Evaluate a model stored in a specified path against a given dataset

Parameters

- **dataset** (BaseDataset | *None*) – the dataset to evaluate against
- **model_path** (*str* | *None*) – the path to the model on disk

Returns

evaluate_model(*dataloader*, *criterion=None*, *description='testing on validation set'*)

Evaluates the current model against the specified dataloader for the specified metrics :param dataloader: The dataloader to evaluate against :param metrics: list of metric keys to calculate :criterion: Criterion to calculate loss :description: What to show in the progress bar :return: tuple of (metrics, y_true, y_pred)

predict(*dataset=None*, *description='running prediction'*)

Predicts using a model against for a specified dataset

Returns
tuple of (y_true, y_pred, y_pred_probs)

Return type
tuple

Parameters
dataset (BaseDataset | *None*) –

predict_image(*image=None*, *labels=None*, *data_transforms=None*, *description='running prediction for single image'*)

Predicts using a model against for a specified image

Returns
Plot containing the image and the predictions.

Return type
matplotlib.figure.Figure

predict_masks(*image=None*, *labels=None*, *data_transforms=None*, *description='running prediction for single image'*)

Predicts using a model against for a specified image

Returns
Plot of the predicted masks

Return type
matplotlib.figure.Figure

detect_objects(*image=None*, *labels=None*, *data_transforms=None*, *description='running object detection for single image'*)

Predicts using a model against for a specified image

Returns
Plots the image with the object boundaries.

Return type
matplotlib.figure.Figure

predict_output_per_batch(*dataloader*, *description*)

Run predictions on a dataloader and return inputs, outputs, labels per batch

forward(**input*, ***kwargs*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

get_predicted(*outputs*, *threshold=None*)

Gets the output from the model and return the predictions :return: Tuple in the format (probabilities, predicted classes/labels) :rtype: tuple

report(*labels*, *dataset_name*, *running_metrics*, ***kwargs*)

The report we want to generate for the model

log_metrics(*output*, *labels*, *tag='train'*, *writer=None*, *epoch=0*)

Log the calculated metrics

allocate_device(*opts=None*)

Put the model on CPU or GPU

Returns
Return the model on CPU or GPU.

Return type
nn.Module

save_model(*model_directory*, *epoch*, *optimizer*, *loss*, *start*, *run_id*)

Saves the model on disk :param model_directory: directory to save the model :param epoch: Epoch number of checkpoint :param optimizer: Optimizer used :param loss: Criterion used :param start: Start time of training :param run_id: Run id of the model

extract_features(**input*, ***kwargs*)

Abstract for trim the model to extract feature. Extending classes should override this method.

Returns
Instance of the model architecture

Return type
nn.Module

load_model(*file_path*, *optimizer=None*)

Loads a model from a checkpoint

load_optimizer()

Load the optimizer

load_criterion()

Load the loss function

load_lr_scheduler(*optimizer*)

train_model(*train_dataset*, *epochs=100*, *model_directory=None*, *save_epochs=10*, *iterations_log=100*, *resume_model=None*, *val_dataset=None*, *run_id=None*, ***kwargs*)

Main method that trains the model.

Parameters

- **train_dataset** (BaseDataset) – Dataset to train the model
- **epochs** (*int, optional*) – Number of epochs for training, defaults to 100
- **model_directory** (*str, optional*) – Directory where the model checkpoints will be saved, defaults to None
- **save_epochs** (*int, optional*) – Number of epochs to save a checkpoint of the model, defaults to 10
- **iterations_log** (*int, optional*) – The number of iterations to pass before logging the system state, defaults to 100
- **resume_model** (*str, optional*) – Boolean indicating whether to resume an already traind model or not, defaults to None
- **val_dataset** (BaseDataset, *optional*) – Dataset used for validation, defaults to None
- **run_id** (*str, optional*) – Optional run id to identify the experiment, defaults to None

Returns
Return the loss of the model

train_and_evaluate_model(*train_dataset*, *epochs=100*, *model_directory=None*, *save_epochs=10*, *iterations_log=100*, *resume_model=None*, *val_dataset=None*, *run_id=None*, ***kwargs*)

Method that trains and evaluates the model.

Parameters

- **train_dataset** (BaseDataset) – Dataset to train the model
- **epochs** (*int, optional*) – Number of epochs for training, defaults to 100
- **model_directory** (*str, optional*) – Model directory where the model checkpoints will be saved, defaults to None
- **save_epochs** (*int, optional*) – Number of epochs to save a checkpoint of the model, defaults to 10
- **iterations_log** (*int, optional*) – Number of iterations to pass before logging the system state, defaults to 100
- **resume_model** (*str, optional*) – Boolean indicating whether to resume an already traind model or not, defaults to None
- **val_dataset** (BaseDataset, *optional*) – Dataset used for validation, defaults to None
- **run_id** (*str, optional*) – Run id to identify the experiment, defaults to None

Returns
Loss of the model

training: bool

aitlas.base.object_detection module

class aitlas.base.object_detection.**BaseObjectDetection**(*config*)

Bases: *BaseModel*

This class extends the functionality of the BaseModel class by adding object detection specific functionality.

BaseModel constructor

Parameters

config (*Config, optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *BaseObjectDetectionSchema*

log_loss = True

get_predicted(*outputs*, *threshold=0.3*)

Get predicted objects from the model outputs.

Parameters

- **outputs** (*torch.Tensor*) – Model outputs with shape (batch_size, num_classes).
- **threshold** (*float, optional*) – The threshold for classification, defaults to None.

Returns

List of dictionaries containing the predicted bounding boxes, scores and labels.

Return type

list

load_optimizer()

Load the optimizer

load_criterion()

Load the loss function

load_lr_scheduler(*optimizer*)

Load the learning rate scheduler

train_epoch(*epoch*, *dataloader*, *optimizer*, *criterion*, *iterations_log*)

Train the model for a single epoch.

Parameters

- **epoch** – The current epoch number.
- **dataloader** – The data loader for the training set.
- **optimizer** – The optimizer.
- **criterion** – The loss function.
- **iterations_log** – The number of iterations after which to log the loss.

Returns

The average loss over the entire epoch.

Return type

float

predict_output_per_batch(*dataloader*, *description*)

Run predictions on a dataloader and return inputs, outputs, targets per batch

Parameters

- **dataloader** (*aitlas.base.BaseDataLoader*) – Data loader for the prediction set.
- **description** (*str*) – Description of the task for logging purposes.

Yield
Yields a tuple of (inputs, outputs, targets)

Return type
tuple

evaluate_model(*dataloader*, *criterion=None*, *description='testing on validation set'*)

Method used to evaluate the model on a validation set.

Parameters

- **dataloader** (*aitlas.base.BaseDataLoader*) – Data loader for the validation set.
- **criterion** (*_type_, optional*) – The loss function, defaults to None.
- **description** (*str, optional*) – Description of the task for logging purposes, defaults to "testing on validation set"

Returns
Returns a MAP score of the evaluation on the model.

Return type
float

training: bool

aitlas.base.schemas module

class aitlas.base.schemas.**BaseDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Schema for configuring a base dataset.

Parameters

- **batch_size** (*int, optional*) – Batch size for the dataset. Default is 64.
- **shuffle** (*bool, optional*) – Flag indicating whether to shuffle the dataset. Default is True.
- **num_workers** (*int, optional*) – Number of workers to use for data loading. Default is 4.
- **pin_memory** (*bool, optional*) – Flag indicating whether to use page-locked memory. Default is False.
- **transforms** (*List[str], optional*) – Classes to run transformations over the input data.
- **target_transforms** (*List[str], optional*) – Classes to run transformations over the target data.
- **joint_transforms** (*List[str], optional*) – Classes to run transformations over the input and target data.

- **labels** (*List[str], optional*) – Labels for the dataset.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.base.schemas.**BaseModelSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Schema for configuring a base model.

Parameters

- **num_classes** (*int, optional*) – Number of classes for the model. Default is 2.
- **use_cuda** (*bool, optional*) – Flag indicating whether to use CUDA if available. Default is True.
- **metrics** (*List[str], optional*) – Metrics to calculate during training and evaluation. Default is ['f1_score'].
- **weights** (*List[float], optional*) – Class weights to apply for the loss function. Default is None.
- **rank** (*int, optional*) – Rank value for distributed data processing. Default is 0.
- **use_ddp** (*bool, optional*) – Flag indicating whether to turn on distributed data processing. Default is False.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.base.schemas.**BaseClassifierSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseModelSchema*

Schema for configuring a base classifier.

Parameters

- **learning_rate** (*float, optional*) – Learning rate used in training. Default is 0.01.
- **weight_decay** (*float, optional*) – Weight decay used in training. Default is 0.0.
- **pretrained** (*bool, optional*) – Flag indicating whether to use a pretrained model. Default is True.
- **local_model_path** (*str, optional*) – Local path of the pretrained model. Default is None.
- **threshold** (*float, optional*) – Prediction threshold if needed. Default is 0.5.
- **freeze** (*bool, optional*) – Flag indicating whether to freeze all layers except for the classifier layer(s). Default is False.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.base.schemas.**BaseSegmentationClassifierSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a base segmentation classifier.

Parameters

- **metrics** (*List[str], optional*) – Classes of metrics you want to calculate during training and evaluation. Default is ['iou', 'f1_score', 'accuracy'].
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –

- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.base.schemas.**BaseObjectDetectionSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a base object detection model.

Parameters

- **metrics** (*List[str], optional*) – Classes of metrics you want to calculate during training and evaluation. Default is ['map'].
- **step_size** (*int, optional*) – Step size for the learning rate scheduler. Default is 15.
- **gamma** (*float, optional*) – Gamma (multiplier) for the learning rate scheduler. Default is 0.1.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.base.schemas.**BaseTransformsSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: `Schema`

Parameters

- **only** (*types.StrSequenceOrSet | None*) –

- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

aitlas.base.segmentation module

class aitlas.base.segmentation.**BaseSegmentationClassifier**(*config*)

Bases: *BaseModel*

Base class for a segmentation classifier.

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *BaseSegmentationClassifierSchema*

get_predicted(*outputs*, *threshold=None*)

Get predicted classes from the model outputs.

Parameters

- **outputs** (*torch.Tensor*) – Model outputs with shape (batch_size, num_classes).
- **threshold** (*float, optional*) – The threshold for classification, defaults to None.

Returns
tuple containing the probabilities and predicted classes

Return type
tuple

load_optimizer()

Load the optimizer

load_criterion()

Load the loss function

load_lr_scheduler(*optimizer*)

Load the learning rate scheduler

training: bool

aitlas.base.tasks module

class aitlas.base.tasks.**BaseTask**(*model*, *config*)

Bases: *Configurable*

static create_dataset(*dataset_config*)

Builds the input dataset using the provided configuration.

generate_task_id()

Generates a task ID

run()

Runs the task.

aitlas.base.transforms module

Base class for implementing configurable transformations

aitlas.base.transforms.**load_transforms**(*class_names*, *config*)

Loads transformation classes and make a composition of them

class aitlas.base.transforms.**BaseTransforms**(**args*, ***kwargs*)

Bases: object

Base class for implementing configurable transformations

schema

alias of *BaseTransformsSchema*

configurables = None

aitlas.base.visualizations module

Base class for implementing visualizations.

class aitlas.base.visualizations.**BaseVisualization**(*cm*, *labels*, *file*, ***kwargs*)

Bases: object

Base class for visualizations

plot()

class aitlas.base.visualizations.**BaseDetailedVisualization**(*y_true*, *y_pred*, *y_prob*, *labels*, *file*, ***kwargs*)

Bases: *BaseVisualization*

Base class for visualizations

plot()

5.2 Datasets module

5.2.1 Datasets package

aitlas.datasets.aid module

class aitlas.datasets.aid.**AIDDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://1drv.ms/u/s!AthY3vMZmuxChNR0Co7QHpJ56M-SvQ'

labels = ['Airport', 'BareLand', 'BaseballField', 'Beach', 'Bridge', 'Center', 'Church', 'Commercial', 'DenseResidential', 'Desert', 'Farmland', 'Forest', 'Industrial', 'Meadow', 'MediumResidential', 'Mountain', 'Park', 'Parking', 'Playground', 'Pond', 'Port', 'RailwayStation', 'Resort', 'River', 'School', 'SparseResidential', 'Square', 'Stadium', 'StorageTanks', 'Viaduct']

name = 'AID dataset'

aitlas.datasets.aid_multilabel module

class aitlas.datasets.aid_multilabel.**AIDMultiLabelDataset**(*config*)

Bases: *MultiLabelClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/Hua-YS/AID-Multilabel-Dataset'

labels = ['airplane', 'bare-soil', 'buildings', 'cars', 'chaparral', 'court', 'dock', 'field', 'grass', 'mobile-home', 'pavement', 'sand', 'sea', 'ship', 'tanks', 'trees', 'water']

name = 'AID multilabel dataset'

aitlas.datasets.airs module

class aitlas.datasets.airs.**AIRSDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://www.airs-dataset.com/'

labels = ['Background', 'Roof']

color_mapping = [[0, 0, 0], [200, 200, 200]]

name = 'AIRS'

aitlas.datasets.amazon_rainforest module

class aitlas.datasets.amazon_rainforest.**AmazonRainforestDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://zenodo.org/record/3233081#.YTYm_44zaUk'

labels = ['Background', 'Forest']

color_mapping = [[0, 0, 0], [0, 255, 0]]

name = 'Amazon Rainforest'

load_dataset(*data_dir*, *csv_file=None*)

aitlas.datasets.big_earth_net module

aitlas.datasets.big_earth_net.**interp_band**(*bands*, *img10_shape=[120, 120]*)

https://github.com/lanha/DSen2/blob/master/utils/patches.py

aitlas.datasets.big_earth_net.**parse_json_labels**(*f_j_path*)

parse meta-data json file for big earth to get image labels

Parameters
f_j_path (*str*) – json file path

Returns
list of labels

Return type
list

aitlas.datasets.big_earth_net.**update_json_labels**(*f_j_path*, *BigEarthNet_19_labels*)

aitlas.datasets.big_earth_net.**loads_pickle**(*buf*)

Parameters
buf (*bytes-like object*) – the output of *dumps*

Returns
object

aitlas.datasets.big_earth_net.**dumps_pickle**(*obj*)

Serialize an object. :param obj: object to be serialized :type obj: bytes-like object :return: Implementation-dependent bytes-like object

aitlas.datasets.big_earth_net.**cls2multihot**(*cls_vec*, *label_indices*)

class aitlas.datasets.big_earth_net.**BigEarthNetDataset**(*config*)

Bases: *BaseDataset*

BigEarthNet dataset adaptation

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema
alias of *BigEarthNetSchema*

name = 'Big Earth Net'

get_labels()
Implement this if you want to return the complete set of labels of the dataset

load_patches()

get_item_name(*index*)

show_image(*index*)
Implement this if you want to return an image with a given index from the dataset

save_image(*index*)

show_batch(*size*, *show_title=True*)
Implement this if you want to return a random batch of images from the dataset

data_distribution_table()
Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()
Implement this if you want to return the label distribution of the dataset as a barchart

labels_stats()

prepare()
Implement if something needs to happen to the dataset after object creation

process_to_lmdb()

class aitlas.datasets.big_earth_net.**PrepBigEarthNetDataset**(*data_dir=None*, *patch_names_list=None*, *label_indices=None*)

Bases: Dataset

aitlas.datasets.brazilian_coffee_scenes module

class aitlas.datasets.brazilian_coffee_scenes.**BrazilianCoffeeScenesDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

```
    url = 'http://www.patreo.dcc.ufmg.br/wp-content/uploads/2017/11/
    brazilian_coffee_dataset.zip'

    labels = ['coffee', 'noncoffee']

    name = 'Brazilian Coffee Scenes dataset'
```

aitlas.datasets.brazilian_coffee_scenes.**prepare**(*root*)

aitlas.datasets.breizhcrops module

BreizhCrops - a crop type classification dataset

Note: Adapted from: https://github.com/dl4sits/BreizhCrops ; Original implementation of BreizhCrops dataset: https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/datasets/breizhcrops.py

---

class aitlas.datasets.breizhcrops.**DownloadProgressBar**(*_, **__)

Bases: tqdm

Parameters

- **iterable** (*iterable, optional*) – Iterable to decorate with a progressbar. Leave blank to manually manage the updates.
- **desc** (*str, optional*) – Prefix for the progressbar.
- **total** (*int or float, optional*) – The number of expected iterations. If unspecified, len(iterable) is used if possible. If float("inf") or as a last resort, only basic progress statistics are displayed (no ETA, no progressbar). If *gui* is True and this parameter needs subsequent updating, specify an initial arbitrary large positive number, e.g. 9e9.
- **leave** (*bool, optional*) – If [default: True], keeps all traces of the progressbar upon termination of iteration. If *None*, will leave only if *position* is *0*.
- **file** (*io.TextIOWrapper* or *io.StringIO*, optional) – Specifies where to output the progress messages (default: sys.stderr). Uses *file.write(str)* and *file.flush()* methods. For encoding, see *write_bytes*.
- **ncols** (*int, optional*) – The width of the entire output message. If specified, dynamically resizes the progressbar to stay within this bound. If unspecified, attempts to use environment width. The fallback is a meter width of 10 and no limit for the counter and statistics. If 0, will not print any meter (only stats).
- **mininterval** (*float, optional*) – Minimum progress display update interval [default: 0.1] seconds.
- **maxinterval** (*float, optional*) – Maximum progress display update interval [default: 10] seconds. Automatically adjusts *miniters* to correspond to *mininterval* after long display update lag. Only works if *dynamic_miniters* or monitor thread is enabled.
- **miniters** (*int or float, optional*) – Minimum progress display update interval, in iterations. If 0 and *dynamic_miniters*, will automatically adjust to equal *mininterval* (more CPU efficient, good for tight loops). If > 0, will skip display of specified number of iterations. Tweak this and *mininterval* to get very efficient loops. If your progress is erratic with both fast and slow iterations (network, skipping items, etc) you should set miniters=1.

- **ascii** (*bool or str, optional*) – If unspecified or False, use unicode (smooth blocks) to fill the meter. The fallback is to use ASCII characters " 123456789#".
- **disable** (*bool, optional*) – Whether to disable the entire progressbar wrapper [default: False]. If set to None, disable on non-TTY.
- **unit** (*str, optional*) – String that will be used to define the unit of each iteration [default: it].
- **unit_scale** (*bool or int or float, optional*) – If 1 or True, the number of iterations will be reduced/scaled automatically and a metric prefix following the International System of Units standard will be added (kilo, mega, etc.) [default: False]. If any other non-zero number, will scale *total* and *n*.
- **dynamic_ncols** (*bool, optional*) – If set, constantly alters *ncols* and *nrows* to the environment (allowing for window resizes) [default: False].
- **smoothing** (*float, optional*) – Exponential moving average smoothing factor for speed estimates (ignored in GUI mode). Ranges from 0 (average speed) to 1 (current/instantaneous speed) [default: 0.3].
- **bar_format** (*str, optional*) – Specify a custom bar string formatting. May impact performance. [default: '{l_bar}{bar}{r_bar}'], where l_bar='{desc}: {percentage:3.0f}%|' and r_bar='| {n_fmt}/{total_fmt} [{elapsed}<{remaining}, '

  '{rate_fmt}{postfix}]'

  Possible vars: l_bar, bar, r_bar, n, n_fmt, total, total_fmt,
  percentage, elapsed, elapsed_s, ncols, nrows, desc, unit, rate, rate_fmt, rate_noinv, rate_noinv_fmt, rate_inv, rate_inv_fmt, postfix, unit_divisor, remaining, remaining_s, eta.

  Note that a trailing ": " is automatically removed after {desc} if the latter is empty.
- **initial** (*int or float, optional*) – The initial counter value. Useful when restarting a progress bar [default: 0]. If using float, consider specifying *{n:.3f}* or similar in *bar_format*, or specifying *unit_scale*.
- **position** (*int, optional*) – Specify the line offset to print this bar (starting from 0) Automatic if unspecified. Useful to manage multiple bars at once (eg, from threads).
- **postfix** (dict or *, optional) – Specify additional stats to display at the end of the bar. Calls *set_postfix(**postfix)* if possible (dict).
- **unit_divisor** (*float, optional*) – [default: 1000], ignored unless *unit_scale* is True.
- **write_bytes** (*bool, optional*) – If (default: None) and *file* is unspecified, bytes will be written in Python 2. If *True* will also write bytes. In all other cases will default to unicode.
- **lock_args** (*tuple, optional*) – Passed to *refresh* for intermediate output (initialisation, iterating, and updating).
- **nrows** (*int, optional*) – The screen height. If specified, hides nested bars outside this bound. If unspecified, attempts to use environment height. The fallback is 20.
- **colour** (*str, optional*) – Bar colour (e.g. 'green', '#00ff00').

- **delay** (*float, optional*) – Don't display until [default: 0] seconds have elapsed.
- **gui** (*bool, optional*) – WARNING: internal parameter - do not use. Use tqdm.gui.tqdm(...) instead. If set, will attempt to use matplotlib animations for a graphical output [default: False].

Returns
out

Return type
decorated iterator.

update_to(*b=1*, *bsize=1*, *tsize=None*)

aitlas.datasets.breizhcrops.**download_file**(*url*, *output_path*, *overwrite=False*)

aitlas.datasets.breizhcrops.**unzip**(*zipfile_path*, *target_dir*)

aitlas.datasets.breizhcrops.**untar**(*filepath*)

class aitlas.datasets.breizhcrops.**BreizhCropsDataset**(*config*)

Bases: *CropsDataset*

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema
alias of *BreizhCropsSchema*

preprocess()

get_labels()
Implement this if you want to return the complete set of labels of the dataset

data_distribution_table()
Implement this if you want to return the label distribution of the dataset

parcel_distribution_table()

data_distribution_barchart()
Implement this if you want to return the label distribution of the dataset as a barchart

show_samples()
Implement this if you want to return a random samples from the dataset

show_timeseries(*index*)

download_csv_files(*region*)

build_folder_structure(*root*, *year*, *level*, *region*)
Folder structure:

```
<root>
   codes.csv
   classmapping.csv
   <year>
      <region>.shp
      <level>
         <region>.csv
```

(continues on next page)

(continued from previous page)

```
        <region>.h5
        <region>
            <csv>
                123123.csv
                123125.csv
                ...
```

get_fid(*idx*)

download_h5_database(*region*)

write_h5_database_from_csv(*index*, *region*)

get_codes()

load_classmapping(*classmapping*)

get_classes_to_ind(*classmapping*)

keep for now, could be needed to make it compatible with GenericMulticlass

load_raw(*csv_file*)

```
['B1', 'B10', 'B11', 'B12', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B8
↪', 'B8A', 'B9', 'QA10', 'QA20', 'QA60', 'doa', 'label', 'id']
```

load(*csv_file*)

load_culturecode_and_id(*csv_file*)

write_index(*region*)

aitlas.datasets.camvid module

class aitlas.datasets.camvid.**CamVidDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/alexgkendall/SegNet-Tutorial'

labels = ['sky', 'building', 'column_pole', 'road', 'sidewalk', 'tree', 'sign', 'fence', 'car', 'pedestrian', 'byciclist', 'void']

color_mapping = [[255, 127, 127], [255, 191, 127], [255, 255, 127], [191, 255, 127], [127, 255, 127], [127, 255, 191], [127, 255, 255], [127, 191, 255], [127, 127, 255], [191, 127, 255], [255, 127, 255], [255, 127, 191]]

name = 'CamVid'

load_dataset(*data_dir*, *csv_file=None*)

aitlas.datasets.chactun module

class aitlas.datasets.chactun.**ChactunDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

labels = ['Aguada', 'Building', 'Platform']

color_mapping = [[255, 255, 0], [100, 100, 100], [0, 255, 0]]

name = 'Chactun'

load_dataset(*data_dir*, *csv_file=None*)

show_image(*index*, *show_title=True*)

Implement this if you want to return an image with a given index from the dataset

aitlas.datasets.clrs module

class aitlas.datasets.clrs.**CLRSDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/lehaifeng/CLRS'

labels = ['airport', 'bare-land', 'beach', 'bridge', 'commercial', 'desert', 'farmland', 'forest', 'golf-course', 'highway', 'industrial', 'meadow', 'mountain', 'overpass', 'park', 'parking', 'playground', 'port', 'railway', 'railway-station', 'residential', 'river', 'runway', 'stadium', 'storage-tank']

name = 'CLRS dataset'

aitlas.datasets.crops_classification module

class aitlas.datasets.crops_classification.**CropsDataset**(*config*)

Bases: *BaseDataset*

CropsDataset - a crop type classification dataset

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *CropsDatasetSchema*

preprocess()

get_labels()
Implement this if you want to return the complete set of labels of the dataset

data_distribution_table()
Implement this if you want to return the label distribution of the dataset

parcel_distribution_table()

data_distribution_barchart()
Implement this if you want to return the label distribution of the dataset as a barchart

show_samples()
Implement this if you want to return a random samples from the dataset

show_image(*index*)
Implement this if you want to return an image with a given index from the dataset

show_timeseries(*index*)

get_codes()

load_classmapping(*classmapping*)

aitlas.datasets.dfc15_multilabel module

class aitlas.datasets.dfc15_multilabel.**DFC15MultiLabelDataset**(*config*)
Bases: *MultiLabelClassificationDataset*

BaseDataset constructor

Parameters
config (*Config, contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/Hua-YS/DFC15-Multilabel-Dataset'

labels = ['impervious', 'water', 'clutter', 'vegetation', 'building', 'tree', 'boat', 'car']

name = 'DFC15 dataset'

aitlas.datasets.eopatch_crops module

class aitlas.datasets.eopatch_crops.**DownloadProgressBar**(*_, **__)
Bases: tqdm

Parameters

- **iterable** (*iterable, optional*) – Iterable to decorate with a progressbar. Leave blank to manually manage the updates.
- **desc** (*str, optional*) – Prefix for the progressbar.
- **total** (*int or float, optional*) – The number of expected iterations. If unspecified, len(iterable) is used if possible. If float("inf") or as a last resort, only basic progress statistics are displayed (no ETA, no progressbar). If *gui* is True and this parameter needs subsequent updating, specify an initial arbitrary large positive number, e.g. 9e9.

- **leave** (*bool, optional*) – If [default: True], keeps all traces of the progressbar upon termination of iteration. If *None*, will leave only if *position* is *0*.
- **file** (*io.TextIOWrapper* or *io.StringIO*, optional) – Specifies where to output the progress messages (default: sys.stderr). Uses *file.write(str)* and *file.flush()* methods. For encoding, see *write_bytes*.
- **ncols** (*int, optional*) – The width of the entire output message. If specified, dynamically resizes the progressbar to stay within this bound. If unspecified, attempts to use environment width. The fallback is a meter width of 10 and no limit for the counter and statistics. If 0, will not print any meter (only stats).
- **mininterval** (*float, optional*) – Minimum progress display update interval [default: 0.1] seconds.
- **maxinterval** (*float, optional*) – Maximum progress display update interval [default: 10] seconds. Automatically adjusts *miniters* to correspond to *mininterval* after long display update lag. Only works if *dynamic_miniters* or monitor thread is enabled.
- **miniters** (*int or float, optional*) – Minimum progress display update interval, in iterations. If 0 and *dynamic_miniters*, will automatically adjust to equal *mininterval* (more CPU efficient, good for tight loops). If > 0, will skip display of specified number of iterations. Tweak this and *mininterval* to get very efficient loops. If your progress is erratic with both fast and slow iterations (network, skipping items, etc) you should set miniters=1.
- **ascii** (*bool or str, optional*) – If unspecified or False, use unicode (smooth blocks) to fill the meter. The fallback is to use ASCII characters ” 123456789#”.
- **disable** (*bool, optional*) – Whether to disable the entire progressbar wrapper [default: False]. If set to None, disable on non-TTY.
- **unit** (*str, optional*) – String that will be used to define the unit of each iteration [default: it].
- **unit_scale** (*bool or int or float, optional*) – If 1 or True, the number of iterations will be reduced/scaled automatically and a metric prefix following the International System of Units standard will be added (kilo, mega, etc.) [default: False]. If any other non-zero number, will scale *total* and *n*.
- **dynamic_ncols** (*bool, optional*) – If set, constantly alters *ncols* and *nrows* to the environment (allowing for window resizes) [default: False].
- **smoothing** (*float, optional*) – Exponential moving average smoothing factor for speed estimates (ignored in GUI mode). Ranges from 0 (average speed) to 1 (current/instantaneous speed) [default: 0.3].
- **bar_format** (*str, optional*) – Specify a custom bar string formatting. May impact performance. [default: ‘{l_bar}{bar}{r_bar}’], where l_bar=’{desc}: {percentage:3.0f}%|’ and r_bar=’| {n_fmt}/{total_fmt} [{elapsed}<{remaining}, ‘

  ’{rate_fmt}{postfix}]’

  Possible vars: l_bar, bar, r_bar, n, n_fmt, total, total_fmt,
  percentage, elapsed, elapsed_s, ncols, nrows, desc, unit, rate, rate_fmt, rate_noinv, rate_noinv_fmt, rate_inv, rate_inv_fmt, postfix, unit_divisor, remaining, remaining_s, eta.

Note that a trailing ": " is automatically removed after {desc} if the latter is empty.

- **initial** (*int or float, optional*) – The initial counter value. Useful when restarting a progress bar [default: 0]. If using float, consider specifying *{n:.3f}* or similar in *bar_format*, or specifying *unit_scale*.
- **position** (*int, optional*) – Specify the line offset to print this bar (starting from 0) Automatic if unspecified. Useful to manage multiple bars at once (eg, from threads).
- **postfix** (dict or *, optional) – Specify additional stats to display at the end of the bar. Calls *set_postfix(**postfix)* if possible (dict).
- **unit_divisor** (*float, optional*) – [default: 1000], ignored unless *unit_scale* is True.
- **write_bytes** (*bool, optional*) – If (default: None) and *file* is unspecified, bytes will be written in Python 2. If *True* will also write bytes. In all other cases will default to unicode.
- **lock_args** (*tuple, optional*) – Passed to *refresh* for intermediate output (initialisation, iterating, and updating).
- **nrows** (*int, optional*) – The screen height. If specified, hides nested bars outside this bound. If unspecified, attempts to use environment height. The fallback is 20.
- **colour** (*str, optional*) – Bar colour (e.g. 'green', '#00ff00').
- **delay** (*float, optional*) – Don't display until [default: 0] seconds have elapsed.
- **gui** (*bool, optional*) – WARNING: internal parameter - do not use. Use tqdm.gui.tqdm(...) instead. If set, will attempt to use matplotlib animations for a graphical output [default: False].

Returns
out

Return type
decorated iterator.

update_to(*b=1*, *bsize=1*, *tsize=None*)

aitlas.datasets.eopatch_crops.**download_file**(*url*, *output_path*, *overwrite=False*)

class aitlas.datasets.eopatch_crops.**EOPatchCrops**(*config*)

Bases: *CropsDataset*

EOPatchCrops - a crop type classification dataset

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

preprocess()

split()

write_index()

aitlas.datasets.eurosat module

class aitlas.datasets.eurosat.**EurosatDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/phelber/EuroSAT'

labels = ['AnnualCrop', 'Forest', 'HerbaceousVegetation', 'Highway', 'Industrial', 'Pasture', 'PermanentCrop', 'Residential', 'River', 'SeaLake']

name = 'EuroSAT dataset'

aitlas.datasets.inria module

class aitlas.datasets.inria.**InriaDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://project.inria.fr/aerialimagelabeling/'

labels = ['Background', 'Buildings']

color_mapping = [[0, 0, 0], [255, 255, 255]]

name = 'Inria'

aitlas.datasets.landcover_ai module

class aitlas.datasets.landcover_ai.**LandCoverAiDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://landcover.ai.linuxpolska.com/'

labels = ['Background', 'Buildings', 'Woodlands', 'Water', 'Road']

color_mapping = [[255, 255, 0], [0, 0, 0], [0, 255, 0], [0, 0, 255], [200, 200, 200]]

name = 'Landcover AI'

aitlas.datasets.landcover_ai.**split_images**(*imgs_dir*, *masks_dir*, *output_dir*)

aitlas.datasets.massachusetts_buildings module

class aitlas.datasets.massachusetts_buildings.**MassachusettsBuildingsDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://www.cs.toronto.edu/~vmnih/data/'

labels = ['Background', 'Buildings']

color_mapping = [[0, 0, 0], [255, 0, 0]]

name = 'Massachusetts Buildings'

aitlas.datasets.massachusetts_roads module

class aitlas.datasets.massachusetts_roads.**MassachusettsRoadsDataset**(*config*)

Bases: *SemanticSegmentationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://www.cs.toronto.edu/~vmnih/data/'

labels = ['Background', 'Roads']

color_mapping = [[0, 0, 0], [200, 200, 200]]

name = 'Massachusetts Roads'

aitlas.datasets.mlrs_net module

class aitlas.datasets.mlrs_net.**MLRSNetMultiLabelDataset**(*config*)

Bases: *MultiLabelClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://data.mendeley.com/datasets/7j9bv9vwsx/2'

labels = ['airplane', 'airport', 'bare soil', 'baseball diamond', 'basketball court', 'beach', 'bridge', 'buildings', 'cars', 'cloud', 'containers', 'crosswalk', 'dense residential area', 'desert', 'dock', 'factory', 'field', 'football field', 'forest', 'freeway', 'golf course', 'grass', 'greenhouse', 'gully', 'habor', 'intersection', 'island', 'lake', 'mobile home', 'mountain', 'overpass', 'park', 'parking lot', 'parkway', 'pavement', 'railway', 'railway station', 'river', 'road', 'roundabout', 'runway', 'sand', 'sea', 'ships', 'snow', 'snowberg', 'sparse residential area', 'stadium', 'swimming pool', 'tanks', 'tennis court', 'terrace', 'track', 'trail', 'transmission tower', 'trees', 'water', 'chaparral', 'wetland', 'wind turbine']

name = 'MLRSNet dataset'

aitlas.datasets.mlrs_net.**prepare**(*root_folder*)

aitlas.datasets.multiclass_classification module

class aitlas.datasets.multiclass_classification.**MultiClassClassificationDataset**(*config*)

Bases: *BaseDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *ClassificationDatasetSchema*

get_labels()

Implement this if you want to return the complete set of labels of the dataset

data_distribution_table()

Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()

Implement this if you want to return the label distribution of the dataset as a barchart

show_samples()

Implement this if you want to return a random samples from the dataset

show_image(*index*)

Implement this if you want to return an image with a given index from the dataset

show_batch(*size*, *show_title=True*)

Implement this if you want to return a random batch of images from the dataset

load_dataset()

re_map_labels(*labels_remapping*)

aitlas.datasets.multilabel_classification module

class aitlas.datasets.multilabel_classification.**MultiLabelClassificationDataset**(*config*)

Bases: *BaseDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *ClassificationDatasetSchema*

get_labels()

Implement this if you want to return the complete set of labels of the dataset

data_distribution_table()

Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()

Implement this if you want to return the label distribution of the dataset as a barchart

show_samples()

Implement this if you want to return a random samples from the dataset

show_image(*index*)

Implement this if you want to return an image with a given index from the dataset

show_batch(*size*, *show_title=True*)

Implement this if you want to return a random batch of images from the dataset

load_dataset(*data_dir*, *csv_file*)

labels_stats()

re_map_labels(*labels_remapping*, *map_size*)

aitlas.datasets.npz module

class aitlas.datasets.npz.**NpzDataset**(*config*)

Bases: *BaseDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *NPZDatasetSchema*

labels = None

get_labels()

Implement this if you want to return the complete set of labels of the dataset

data_distribution_table()

Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()
Implement this if you want to return the label distribution of the dataset as a barchart

show_samples()
Implement this if you want to return a random samples from the dataset

show_image(*index*)
Implement this if you want to return an image with a given index from the dataset

show_batch(*size*, *show_title=True*)
Implement this if you want to return a random batch of images from the dataset

load_dataset()

aitlas.datasets.object_detection module

class aitlas.datasets.object_detection.**BaseObjectDetectionDataset**(*config*)
Bases: *BaseDataset*

Base object detection dataset class

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

name = 'Object Detection Dataset'

dataloader()
Create and return a dataloader for the dataset

apply_transformations(*image*, *target*)

get_labels()
Implement this if you want to return the complete set of labels of the dataset

show_image(*index*, *show_title=False*)
Implement this if you want to return an image with a given index from the dataset

show_batch(*size*, *show_labels=False*)
Implement this if you want to return a random batch of images from the dataset

class aitlas.datasets.object_detection.**ObjectDetectionPascalDataset**(*config*)
Bases: *BaseObjectDetectionDataset*

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema
alias of *ObjectDetectionPascalDatasetSchema*

labels = [None]

load_dataset(*imageset_file*, *data_dir*)

data_distribution_table()
Implement this if you want to return the label distribution of the dataset

data_distribution_barchart(*show_title=True*)

Implement this if you want to return the label distribution of the dataset as a barchart

class aitlas.datasets.object_detection.**ObjectDetectionCocoDataset**(*config*)

Bases: *BaseObjectDetectionDataset*

This is a skeleton object detection dataset following the Coco format

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *ObjectDetectionCocoDatasetSchema*

data_distribution_table()

Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()

Implement this if you want to return the label distribution of the dataset as a barchart

show_samples()

Implement this if you want to return a random samples from the dataset

load_dataset(*data_dir=None*, *json_file=None*)

aitlas.datasets.optimal_31 module

class aitlas.datasets.optimal_31.**Optimal31Dataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://drive.google.com/file/d/
1Fk9a0DW8UyyQsR8dP2Qdakmr69NVBhq9/view'

labels = ['airplane', 'airport', 'baseball_diamond',
'basketball_court', 'beach', 'bridge', 'chaparral', 'church',
'circular_farmland', 'commercial_area', 'dense_residential', 'desert',
'forest', 'freeway', 'golf_course', 'ground_track_field', 'harbor',
'industrial_area', 'intersection', 'island', 'lake', 'meadow',
'medium_residential', 'mobile_home_park', 'mountain', 'overpass',
'parking_lot', 'railway', 'rectangular_farmland', 'roundabout',
'runway']

name = 'Optimal31 dataset'

aitlas.datasets.pattern_net module

class aitlas.datasets.pattern_net.**PatternNetDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://arxiv.org/abs/1706.03424'

labels = ['airplane', 'baseball_field', 'basketball_court', 'beach', 'bridge', 'cemetery', 'chaparral', 'christmas_tree_farm', 'closed_road', 'coastal_mansion', 'crosswalk', 'dense_residential', 'ferry_terminal', 'football_field', 'forest', 'freeway', 'golf_course', 'harbor', 'intersection', 'mobile_home_park', 'nursing_home', 'oil_gas_field', 'oil_well', 'overpass', 'parking_lot', 'parking_space', 'railway', 'river', 'runway', 'runway_marking', 'shipping_yard', 'solar_panel', 'sparse_residential', 'storage_tank', 'swimming_pool', 'tennis_court', 'transformer_station', 'wastewater_treatment_plant']

name = 'PatternNet dataset'

aitlas.datasets.planet_uas module

class aitlas.datasets.planet_uas.**PlanetUASMultiLabelDataset**(*config*)

Bases: *MultiLabelClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://www.kaggle.com/c/planet-understanding-the-amazon-from-space/overview'

labels = ['haze', 'primary', 'agriculture', 'clear', 'water', 'habitation', 'road', 'cultivation', 'slash_burn', 'cloudy', 'partly_cloudy', 'conventional_mine', 'bare_ground', 'artisinal_mine', 'blooming', 'selective_logging', 'blow_down']

name = 'Planet UAS multilabel dataset'

aitlas.datasets.planet_uas.**prepare**(*csv_train_file*)

aitlas.datasets.planet_uas.**kaggle_format**(*csv_file_path*, *output_file*, *threshold*)

aitlas.datasets.resisc45 module

class aitlas.datasets.resisc45.**Resisc45Dataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://www.tensorflow.org/datasets/catalog/resisc45'

labels = ['airplane', 'airport', 'baseball_diamond', 'basketball_court', 'beach', 'bridge', 'chaparral', 'church', 'circular_farmland', 'cloud', 'commercial_area', 'dense_residential', 'desert', 'forest', 'freeway', 'golf_course', 'ground_track_field', 'harbor', 'industrial_area', 'intersection', 'island', 'lake', 'meadow', 'medium_residential', 'mobile_home_park', 'mountain', 'overpass', 'palace', 'parking_lot', 'railway', 'railway_station', 'rectangular_farmland', 'river', 'roundabout', 'runway', 'sea_ice', 'ship', 'snowberg', 'sparse_residential', 'stadium', 'storage_tank', 'tennis_court', 'terrace', 'thermal_power_station', 'wetland']

name = 'RESISC45 dataset'

aitlas.datasets.rsd46_whu module

class aitlas.datasets.rsd46_whu.**RSD46WHUDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/RSIA-LIESMARS-WHU/RSD46-WHU'

labels = ['Airplane', 'Airport', 'Artificial dense forest land', 'Artificial sparse forest land', 'Bare land', 'Basketball court', 'Blue structured factory building', 'Building', 'Construction site', 'Cross river bridge', 'Crossroads', 'Dense tall building', 'Dock', 'Fish pond', 'Footbridge', 'Graff', 'Grassland', 'Low scattered building', 'Lrregular farmland', 'Medium density scattered building', 'Medium density structured building', 'Natural dense forest land', 'Natural sparse forest land', 'Oiltank', 'Overpass', 'Parking lot', 'Plasticgreenhouse', 'Playground', 'Railway', 'Red structured factory building', 'Refinery', 'Regular farmland', 'Scattered blue roof factory building', 'Scattered red roof factory building', 'Sewage plant-type-one', 'Sewage plant-type-two', 'Ship', 'Solar power station', 'Sparse residential area', 'Square', 'Steelsmelter', 'Storage land', 'Tennis court', 'Thermal power plant', 'Vegetable plot', 'Water']

name = 'RSD46-WHU dataset'

aitlas.datasets.rsi_cb256 module

class aitlas.datasets.rsi_cb256.**RSICB256Dataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/lehaifeng/RSI-CB'

labels = ['airplane', 'airport_runway', 'artificial_grassland', 'avenue', 'bare_land', 'bridge', 'city_building', 'coastline', 'container', 'crossroads', 'dam', 'desert', 'dry_farm', 'forest', 'green_farmland', 'highway', 'hirst', 'lakeshore', 'mangrove', 'marina', 'mountain', 'parkinglot', 'pipeline', 'residents', 'river', 'river_protection_forest', 'sandbeach', 'sapling', 'sea', 'shrubwood', 'snow_mountain', 'sparse_forest', 'storage_room', 'stream', 'town']

name = 'RSI-CB256 dataset'

aitlas.datasets.rsscn7 module

class aitlas.datasets.rsscn7.**RSSCN7Dataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxxaW56b3VjbnxneDo1MDYzYWMxOWIwMjRiMWFi'

labels = ['farm_land', 'forest', 'grass_land', 'industrial_region', 'parking_lot', 'residential_region', 'river_lake']

name = 'RSSCN7 dataset'

aitlas.datasets.sat6 module

class aitlas.datasets.sat6.**SAT6Dataset**(*config*)

Bases: *BaseDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *MatDatasetSchema*

url = 'http://csc.lsu.edu/~saikat/deepsat/'

labels = ['buildings', 'barren land', 'trees', 'grassland', 'roads', 'water bodies']

name = 'SAT-6 dataset'

get_labels()
Implement this if you want to return the complete set of labels of the dataset

data_distribution_table()
Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()
Implement this if you want to return the label distribution of the dataset as a barchart

show_image(*index*)
Implement this if you want to return an image with a given index from the dataset

show_batch(*size*, *show_title=True*)
Implement this if you want to return a random batch of images from the dataset

load_dataset(*mat_file*)

re_map_labels(*labels_remapping*)

aitlas.datasets.schemas module

class aitlas.datasets.schemas.**MatDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring a classification dataset given as mat file.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]
Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**NPZDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring a classification dataset given as npz file.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**ClassificationDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring a classification dataset.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**SegmentationDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring a segmentation dataset.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**ObjectDetectionPascalDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring an object detection dataset given in PASCAL VOC format.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –

- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**ObjectDetectionCocoDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring an object detection dataset given in COCO format.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**BigEarthNetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring the BigEarthNet dataset.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –

- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**SpaceNet6DatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring the SpaceNet6 dataset.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**BreizhCropsSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring the BreizhCrops dataset for crop type prediction.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –

- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**CropsDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring dataset for crop type prediction.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.datasets.schemas.**So2SatDatasetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseDatasetSchema*

Schema for configuring the So2Sat dataset.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –

- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

aitlas.datasets.semantic_segmentation module

class aitlas.datasets.semantic_segmentation.**SemanticSegmentationDataset**(*config*)

Bases: *BaseDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema

alias of *SegmentationDatasetSchema*

labels = None

color_mapping = None

name = None

apply_transformations(*image*, *mask*)

load_dataset(*data_dir*, *csv_file=None*)

get_labels()

Implement this if you want to return the complete set of labels of the dataset

data_distribution_table()

Implement this if you want to return the label distribution of the dataset

data_distribution_barchart(*show_title=True*)

Implement this if you want to return the label distribution of the dataset as a barchart

show_image(*index*, *show_title=False*)

Implement this if you want to return an image with a given index from the dataset

aitlas.datasets.siri_whu module

class aitlas.datasets.siri_whu.**SiriWhuDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'http://www.lmars.whu.edu.cn/prof_web/zhongyanfei/e-code.html'

labels = ['agriculture', 'commercial', 'harbor', 'idle_land', 'industrial', 'meadow', 'overpass', 'park', 'pond', 'residential', 'river', 'water']

name = 'SIRI-WHU dataset'

aitlas.datasets.so2sat module

class aitlas.datasets.so2sat.**So2SatDataset**(*config*)

Bases: *BaseDataset*

So2Sat dataset version 2 (contains train, validation and test splits)

So2Sat LCZ42 is a dataset consisting of corresponding synthetic aperture radar and multi-spectral optical image data acquired by the Sentinel-1 and Sentinel-2 remote sensing satellites, and a corresponding local climate zones (LCZ) label. The dataset is distributed over 42 cities across different continents and cultural regions of the world, and comes with a split into fully independent, non-overlapping training, validation, and test sets.

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://dataserv.ub.tum.de/s/m1483140/download?path=%2F&files=testing.h5'

name = 'So2Sat dataset'

schema

alias of *So2SatDatasetSchema*

labels = ['Compact high_rise', 'Compact middle_rise', 'Compact low_rise', 'Open high_rise', 'Open middle_rise', 'Open low_rise', 'Lightweight low_rise', 'Large low_rise', 'Sparsely built', 'Heavy industry', 'Dense trees', 'Scattered trees', 'Bush or scrub', 'Low plants', 'Bare rock or paved', 'Bare soil or sand', 'Water']

get_labels()

Implement this if you want to return the complete set of labels of the dataset

show_image(*index*)

Implement this if you want to return an image with a given index from the dataset

show_samples()
Implement this if you want to return a random samples from the dataset

show_batch(*size*, *show_title=True*)
Implement this if you want to return a random batch of images from the dataset

data_distribution_table()
Implement this if you want to return the label distribution of the dataset

data_distribution_barchart()
Implement this if you want to return the label distribution of the dataset as a barchart

aitlas.datasets.spacenet6 module

aitlas.datasets.spacenet6.**polygon_to_mask**(*poly*, *image_size*)

aitlas.datasets.spacenet6.**process_image**(*image_path*, *segmentation_directory*, *edge_width*, *contact_width*, *gt_buildings_csv*)

Creates and saves the target (ground-truth) segmentation mask for the input image.

Parameters

- **image_path** (*str*) – path to the source image
- **segmentation_directory** (*str*) – path to the destination directory for the segmentation masks
- **edge_width** (*int*) – the width of the edge
- **contact_width** (*int*) – the width of the contact
- **gt_buildings_csv** (*str*) – path to the source ground-truth-buildings csv

class aitlas.datasets.spacenet6.**SpaceNet6Dataset**(*config*)

Bases: *BaseDataset*

SpaceNet6 dataset.

BaseDataset constructor

Parameters
config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

schema
alias of *SpaceNet6DatasetSchema*

load_directory()
Loads the *.tif images from the specified directory.

load_other_folds(*fold*)
Loads all images (and masks) except the ones from this fold.

load_fold(*fold*)
Loads the images from this fold.

labels()

prepare()
Prepares the SpaceNet6 data set for model training and validation by:

1. Creating training segmentation masks from the geojson files

2. Splitting the data set by location, which was shown to be very important for model learning, see: https://github.com/SpaceNetChallenge/SpaceNet_SAR_Buildings_Solutions/blob/master/1-zbigniewwojna/README.md Creates 10 splits of the data set. Each split consists of 10 folds (i.e. further splits) of which 9 are used for training and one for validation/testing (in essence, a cross validation procedure).

aitlas.datasets.uc_merced module

class aitlas.datasets.uc_merced.**UcMercedDataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

labels = ['agricultural', 'airplane', 'baseballdiamond', 'beach', 'buildings', 'chaparral', 'denseresidential', 'forest', 'freeway', 'golfcourse', 'harbor', 'intersection', 'mediumresidential', 'mobilehomepark', 'overpass', 'parkinglot', 'river', 'runway', 'sparseresidential', 'storagetanks', 'tenniscourt']

name = 'UC Merced dataset'

aitlas.datasets.uc_merced_multilabel module

class aitlas.datasets.uc_merced_multilabel.**UcMercedMultiLabelDataset**(*config*)

Bases: *MultiLabelClassificationDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://drive.google.com/file/d/1DtKiauowCB0ykjFe8v0OVvT76rEfOk0v/view'

labels = ['airplane', 'bare-soil', 'buildings', 'cars', 'chaparral', 'court', 'dock', 'field', 'grass', 'mobile-home', 'pavement', 'sand', 'sea', 'ship', 'tanks', 'trees', 'water']

name = 'UC Merced multilabel dataset'

aitlas.datasets.urls module

Contains raw urls to download the data for crop type prediction tasks. TODO Refactor raw csv urls to be more general

aitlas.datasets.whu_rs19 module

class aitlas.datasets.whu_rs19.**WHURS19Dataset**(*config*)

Bases: *MultiClassClassificationDataset*

BaseDataset constructor

Parameters

config (*Config, contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

url = 'https://github.com/CAPTAIN-WHU/BED4RS'

labels = ['Airport', 'Beach', 'Bridge', 'Commercial', 'Desert', 'Farmland', 'footballField', 'Forest', 'Industrial', 'Meadow', 'Mountain', 'Park', 'Parking', 'Pond', 'Port', 'railwayStation', 'Residential', 'River', 'Viaduct']

name = 'WHU-RS19 dataset'

5.3 Transforms module

5.3.1 Transforms module

aitlas.transforms.big_earth_net module

Contains classes for image transformations specific for Big Earth Net dataset.

class aitlas.transforms.big_earth_net.**ResizeToTensorNormalizeRGB**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing, tensor conversion, and normalization to RGB images.

Initialize the class with the given mean and standard deviation for normalization.

Parameters

- **bands10_mean** (*list*) – Mean values for the RGB bands
- **bands10_std** (*list*) – Standard deviation values for the RGB bands

configurables = ['bands10_mean', 'bands10_std']

class aitlas.transforms.big_earth_net.**ToTensorResizeRandomCropFlipHV**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing, tensor conversion, random cropping, and random flipping to images.

class aitlas.transforms.big_earth_net.**ToTensorResizeCenterCrop**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing, tensor conversion, and center cropping to images.

class aitlas.transforms.big_earth_net.**ToTensorResize**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing and tensor conversion to images.

class aitlas.transforms.big_earth_net.**NormalizeAllBands**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies normalization to all bands of the input.

Initialize the class with the given mean and standard deviation for normalization.

Parameters

- **bands10_mean** (*list*) – Mean values for the bands10
- **bands10_std** (*list*) – Standard deviation values for the bands10
- **bands20_mean** (*list*) – Mean values for the bands20
- **bands20_std** (*list*) – Standard deviation values for the bands20

configurables = ['bands10_mean', 'bands10_std', 'bands20_mean', 'bands20_std']

class aitlas.transforms.big_earth_net.**ToTensorAllBands**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class for converting all bands (list) to tensors.

aitlas.transforms.breizhcrops module

Contains classes for image transformations specific for BreizhCrops dataset.

class aitlas.transforms.breizhcrops.**SelectBands**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class used to select and process spectral bands from satellite data.

Parameters

level (*str*) – satellite data level to be processed (“L1C” or “L2A”)

Note: This class requires a level argument at initialization. This should be one of the predefined satellite data levels (“L1C” or “L2A”).

Initialize the SelectBands class by setting the satellite data level.

configurables = ['level']

aitlas.transforms.classification module

Contains classes for image transformations for classification datasets.

class aitlas.transforms.classification.**ResizeRandomCropFlipHVToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing to (256,256), random cropping to size (224,224), random flipping, and tensor conversion to images.

class aitlas.transforms.classification.**ResizeCenterCropFlipHVToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing to (256,256), center cropping to size (224,224), random HV flipping, and tensor conversion to images.

class aitlas.transforms.classification.**ResizeCenterCropToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing to (256,256), center cropping to size (224,224), and tensor conversion to images.

class aitlas.transforms.classification.**Resize1ToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies fixed resizing to (224,224) and tensor conversion to images.

class aitlas.transforms.classification.**GrayToRGB**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that converts grayscale images to RGB format [height, width, channels].

class aitlas.transforms.classification.**ConvertToRGBResizeCenterCropToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that converts an image to RGB format, applies resizing to size (256,256), center cropping to size (224,224), and tensor conversion.

class aitlas.transforms.classification.**RandomFlipHVToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies random flipping and tensor conversion to images.

class aitlas.transforms.classification.**ComplexTransform**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies complex transformations to images and tensor conversion.

The transformations include:

- resizing to (256,256), random cropping to size (224,224),
- random flipping (H and V) with probability 50%,
- random brightness and constrast with probability 75%,
- random blur (motion, median, gaussian, and noise) with probability 70%,
- random distortion (optical, grid, elastic) with probability 70%„
- random CLAHE with probability 70%,
- random HSV shift with probability 50%,

aitlas.transforms.joint_transforms module

Contains joint transforms for images and label masks.

class aitlas.transforms.joint_transforms.**FlipHVRandomRotate**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies flipping, random rotation, and shift-scale-rotation transformations to image and mask pairs.

class aitlas.transforms.joint_transforms.**FlipHVToTensorV2**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing, flipping, and tensor conversion to images with bounding boxes and labels.

class aitlas.transforms.joint_transforms.**ResizeToTensorV2**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing and tensor conversion to images with bounding boxes and labels.

class aitlas.transforms.joint_transforms.**Resize**(**args*, ***kwargs*)

Bases: *BaseTransforms*

A class that applies resizing to images.

aitlas.transforms.object_detection module

aitlas.transforms.segmentation module

Classes and methods for image transformations for segmentation tasks. For semantic segmentation tasks the shape of the input is (N, 3, H, W); The shape of the output/mask is (N, num_classes, H, W), where N is the number of images

class aitlas.transforms.segmentation.**MinMaxNormTranspose**(**args*, ***kwargs*)

Bases: *BaseTransforms*

MinMax Normalization and transposing a given sample.

class aitlas.transforms.segmentation.**Transpose**(**args*, ***kwargs*)

Bases: *BaseTransforms*

Transposes a given sample.

class aitlas.transforms.segmentation.**MinMaxNorm**(**args*, ***kwargs*)

Bases: *BaseTransforms*

MinMax-Normalization of a given sample.

class aitlas.transforms.segmentation.**Pad**(**args*, ***kwargs*)

Bases: *BaseTransforms*

Applies padding to a given sample.

class aitlas.transforms.segmentation.**ColorTransformations**(**args*, ***kwargs*)

Bases: *BaseTransforms*

Applies a set of color transformations to a given sample.

class aitlas.transforms.segmentation.**ResizeToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

Resizes and converts a given sample to a tensor.

class aitlas.transforms.segmentation.**ResizePerChannelToTensor**(**args*, ***kwargs*)

Bases: *BaseTransforms*

aitlas.transforms.spacenet6 module

Classes and methods for image transformations specific for the Spacenet6 dataset.

aitlas.transforms.spacenet6.**saturation**(*img*, *alpha*)

Adjust the saturation of an image.

Parameters

- **img** (*numpy.ndarray*) – input image
- **alpha** (*float*) – saturation factor

Returns
image with adjusted saturation

Return type
numpy.ndarray

aitlas.transforms.spacenet6.**brightness**(*img*, *alpha*)

Adjust the brightness of an image.

Parameters

- **img** (*numpy.ndarray*) – input image
- **alpha** (*float*) – brightness factor

Returns
image with adjusted brightness

Return type
numpy.ndarray

aitlas.transforms.spacenet6.**contrast**(*img*, *alpha*)

Adjust the contrast of an image.

Parameters

- **img** (*numpy.ndarray*) – input image
- **alpha** (*float*) – contrast factor

Returns
image with adjusted contrast

Return type
numpy.ndarray

class aitlas.transforms.spacenet6.**SpaceNet6Transforms**(**args*, ***kwargs*)

Bases: *BaseTransforms*

SpaceNet6 specific image transformations.

5.4 Models module

5.4.1 Models module

aitlas.models.alexnet module

AlexNet model for multiclass and multilabel classification

class aitlas.models.alexnet.**AlexNet**(*config*)

Bases: *BaseMulticlassClassifier*

AlexNet model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.alexnet.html#torchvision.models.alexnet

BaseModel constructor

Parameters
config (*Config, optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'AlexNet'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.alexnet.**AlexNetMultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'AlexNet'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

training: bool

freeze()

aitlas.models.cnn_rnn module

CNNRNN model

class aitlas.models.cnn_rnn.**EncoderCNN**(*embed_size*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*images*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.cnn_rnn.**DecoderRNN**(*embed_size*, *hidden_size*, *num_classes*, *num_layers*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*features*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.cnn_rnn.**CNNRNN**(*config*)

Bases: *BaseMultilabelClassifier*

CNNRNN model implementation.

Note: Based on https://github.com/Lin-Zhipeng/CNN-RNN-A-Unified-Framework-for-Multi-label-Image-C

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *CNNRNNModelSchema*

forward(*inputs*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.convnext module

ConvNeXt tiny model

class aitlas.models.convnext.**ConvNeXtTiny**(*config*)

Bases: *BaseMulticlassClassifier*

ConvNeXtTiny model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.convnext_tiny.html#torchvision.models.convnext_tiny

---

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ConvNeXt tiny'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

freeze()

extract_features()

Remove final layers if we only need to extract features

training: bool

class aitlas.models.convnext.**ConvNeXtTinyMultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ConvNeXt tiny'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

training: bool

freeze()

aitlas.models.deeplabv3 module

DeepLabV3 model

class aitlas.models.deeplabv3.**DeepLabV3**(*config*)

Bases: *BaseSegmentationClassifier*

DeepLabV3 model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.segmentation.deeplabv3_resnet101.html#torchvision.models.segmentation.deeplabv3_resnet101

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.deeplabv3plus module

DeepLabV3Plus model

class aitlas.models.deeplabv3plus.**DeepLabV3Plus**(*config*)

Bases: *BaseSegmentationClassifier*

DeepLabV3Plus model implementation

Note: Based on https://github.com/qubvel/segmentation_models.pytorch

---

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.densenet module

DenseNet161 model for multiclass classification

class aitlas.models.densenet.**DenseNet161**(*config*)

Bases: *BaseMulticlassClassifier*

DenseNet161 model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.densenet161.html#torchvision.models.densenet161

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'DenseNet161'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.densenet.**DenseNet161MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'DenseNet161'

training: bool

forward(*x*)
Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()
Remove final layers if we only need to extract features

freeze()

aitlas.models.efficientnet module

EfficientNetB0 (V1) for image classification

class aitlas.models.efficientnet.**EfficientNetB0**(*config*)
Bases: *BaseMulticlassClassifier*

EfficientNetB0 model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.efficientnet_b0.html#torchvision.models.efficientnet_b0

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetB0'

forward(*x*)
Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

freeze()

extract_features()
Abstract for trim the model to extract feature. Extending classes should override this method.

Returns
Instance of the model architecture

Return type
nn.Module

training: bool

class aitlas.models.efficientnet.**EfficientNetB0MultiLabel**(*config*)
Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetB0'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.efficientnet.**EfficientNetB4**(*config*)

Bases: *BaseMulticlassClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetB4'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

freeze()

extract_features()

Abstract for trim the model to extract feature. Extending classes should override this method.

Returns
Instance of the model architecture

Return type
nn.Module

training: bool

class aitlas.models.efficientnet.**EfficientNetB4MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetB4'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.efficientnet.**EfficientNetB7**(*config*)

Bases: *BaseMulticlassClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetB7'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

freeze()

extract_features()

Abstract for trim the model to extract feature. Extending classes should override this method.

Returns
Instance of the model architecture

Return type
nn.Module

training: bool

class aitlas.models.efficientnet.**EfficientNetB7MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetB7'

training: bool

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

aitlas.models.efficientnet_v2 module

EfficientNetV2 model

class aitlas.models.efficientnet_v2.**EfficientNetV2**(*config*)

Bases: *BaseMulticlassClassifier*

EfficientNetV2 model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.efficientnet_v2_m.html#torchvision.models.efficientnet_v2_m

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetV2'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

class aitlas.models.efficientnet_v2.**EfficientNetV2MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'EfficientNetV2'

training: bool

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

aitlas.models.fasterrcnn module

FasterRCNN model for object detection

class aitlas.models.fasterrcnn.**FasterRCNN**(*config*)

Bases: *BaseObjectDetection*

FasterRCNN model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn_v2.html#torchvision.models.detection.fasterrcnn_resnet50_fpn_v2

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*inputs*, *targets=None*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.fcn module

FCN model for segmentation

class aitlas.models.fcn.**FCN**(*config*)

Bases: *BaseSegmentationClassifier*

FCN model implementation

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.segmentation.fcn_resnet101.html#torchvision.models.segmentation.fcn_resnet101

BaseModel constructor

Parameters

config (*Config, optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.hrnet module

HRNet model for segmentation

class aitlas.models.hrnet.**HRNetModule**(*head*, *pretrained=True*, *higher_res=False*)

Bases: Module

HRNet model implementation

Note: Based on https://github.com/huggingface/pytorch-image-models/tree/main/timm

Pretrained backbone for HRNet. :param head: Output head :type head: nn.Module :param pretrained: If True, uses imagenet pretrained weights :type pretrained: bool :param higher_res: If True, retains higher resolution features :type higher_res: bool

Parameters

- **head** (*Module*) –
- **pretrained** (*bool*) –
- **higher_res** (*bool*) –

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

class aitlas.models.hrnet.**HRNetSegHead**(*nclasses=3*, *higher_res=False*)

Bases: Module

Segmentation head for HRNet. Does not have pretrained weights.

Parameters

- **nclasses** (*int*) – Number of output classes
- **higher_res** (*bool*) – If True, retains higher resolution features

forward(*x*, *yl*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.hrnet.**HRNet**(*config*, *higher_res=False*)

Bases: *BaseSegmentationClassifier*

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.inceptiontime module

InceptionTime model

Note: Original implementation of InceptionTime model https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/models/InceptionTime.py

---

class aitlas.models.inceptiontime.**InceptionTime**(*config*)

Bases: *BaseMulticlassClassifier*

InceptionTime model implementation

Note: Based https://github.com/dl4sits/BreizhCrops

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *InceptionTimeSchema*

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

training: bool

class aitlas.models.inceptiontime.**InceptionModule**(*kernel_size=32*, *num_filters=128*, *residual=True*, *use_bias=False*, *device=device(type='cpu')*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input_tensor*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

aitlas.models.lstm module

LSTM model

Note: Original implementation of LSTM model: https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/models/LongShortTermMemory.py

---

class aitlas.models.lstm.**LSTM**(*config*)

Bases: *BaseMulticlassClassifier*

LSTM model implementation

Note: Based on <https://github.com/dl4sits/BreizhCrops>

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *LSTMSchema*

logits(*x*)

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

training: bool

aitlas.models.mlp_mixer module

MLP-Mixer architecture for image classification.

class aitlas.models.mlp_mixer.**MLPMixer**(*config*)

Bases: *BaseMulticlassClassifier*

MLP mixer multi-class b16_224 model implementation

Note: Based on <https://github.com/huggingface/pytorch-image-models>

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'MLP mixer b16_224'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

class aitlas.models.mlp_mixer.**MLPMixerMultilabel**(*config*)

Bases: *BaseMultilabelClassifier*

MLP mixer multi-label b16_224 model implementation

Note: Based on <https://github.com/huggingface/pytorch-image-models>

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'MLP mixer b16_224'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.msresnet module

MRSResNet model

Note: Adapted from https://github.com/dl4sits/BreizhCrops Original implementation of MSResNet model: https://github.com/geekfeiw/Multi-Scale-1D-ResNet/blob/master/model/multi_scale_ori.py https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/models/MSResNet.py

---

aitlas.models.msresnet.**conv3x3**(*in_planes*, *out_planes*, *stride=1*)

3x3 convolution with padding

aitlas.models.msresnet.**conv5x5**(*in_planes*, *out_planes*, *stride=1*)

aitlas.models.msresnet.**conv7x7**(*in_planes*, *out_planes*, *stride=1*)

class aitlas.models.msresnet.**BasicBlock3x3**(*inplanes3*, *planes*, *stride=1*, *downsample=None*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

expansion = 1

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.msresnet.**BasicBlock5x5**(*inplanes5*, *planes*, *stride=1*, *downsample=None*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

expansion = 1

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.msresnet.**BasicBlock7x7**(*inplanes7*, *planes*, *stride=1*, *downsample=None*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

expansion = 1

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.msresnet.**MSResNet**(*config*)

Bases: *BaseMulticlassClassifier*

MSResNet model implementation

Note: Based on <https://github.com/dl4sits/BreizhCrops>

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *MSResNetSchema*

forward(*x0*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

training: bool

aitlas.models.omniscalecnn module

OmniScaleCNN model implementation

Note: Adapted from https://github.com/dl4sits/BreizhCrops ; Original implementation of OmniScaleCNN model: https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/models/OmniScaleCNN.py

---

class aitlas.models.omniscalecnn.**SampaddingConv1D_BN**(*in_channels*, *out_channels*, *kernel_size*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*X*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

class aitlas.models.omniscalecnn.**build_layer_with_layer_parameter**(*layer_parameters*)

Bases: `Module`

formerly build_layer_with_layer_parameter

Note: layer_parameters format : [in_channels, out_channels, kernel_size, in_channels, out_channels, kernel_size, ..., nlayers]

forward(*X*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

class aitlas.models.omniscalecnn.**OmniScaleCNN**(*config*)

Bases: *BaseMulticlassClassifier*

OmniScaleCNN model implementation

Note: Based on <https://github.com/dl4sits/BreizhCrops>

BaseModel constructor

Parameters

config (*Config, optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *OmniScaleCNNSchema*

forward(*X*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

training: bool

aitlas.models.omniscalecnn.**get_Prime_number_in_a_range**(*start*, *end*)

aitlas.models.omniscalecnn.**get_out_channel_number**(*paramenter_layer*, *in_channel*, *prime_list*)

aitlas.models.omniscalecnn.**generate_layer_parameter_list**(*start*, *end*, *paramenter_number_of_layer_list*, *in_channel=1*)

aitlas.models.resnet module

ResNet50 and ResNet152 models for multi-class and multi-label classification

class aitlas.models.resnet.**ResNet50**(*config*)

Bases: *BaseMulticlassClassifier*

ResNet50 multi-class model implementation.

Note: Based on https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet50.html#torchvision.models.resnet50

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ResNet50'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

freeze()

extract_features()

Remove final layers if we only need to extract features

training: bool

class aitlas.models.resnet.**ResNet152**(*config*)

Bases: *BaseMulticlassClassifier*

ResNet50 multi-label model implementation

Note: Based on <https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet50.html#torchvision.models.resnet50>

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ResNet152'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.resnet.**ResNet50MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ResNet50'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.resnet.**ResNet152MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ResNet152'

training: bool

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

aitlas.models.schemas module

class aitlas.models.schemas.**TransformerModelSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a transformer model.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –

- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**InceptionTimeSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a InceptionTime model.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**LSTMSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a LSTM model.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –

- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**MSResNetSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a MSResNet model.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**TempCNNSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a TempCNN model.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]
Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**StarRNNSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a StarRNN model.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]
Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**OmniScaleCNNSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseClassifierSchema*

Schema for configuring a OmniScaleCNN model.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]
Dictionary mapping field_names -> Field objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**UnsupervisedDeepMulticlassClassifierSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseModelSchema*

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]
Dictionary mapping field_names -> Field objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**UNetEfficientNetModelSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseSegmentationClassifierSchema*

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –

- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.models.schemas.**CNNRNNModelSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseModelSchema*

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

aitlas.models.shallow module

class aitlas.models.shallow.**ShallowCNNNet**(*config*)

Bases: *BaseMulticlassClassifier*

Simlpe shallow multi-class CNN network for testing purposes

BaseModel constructor

Parameters

config (*Config, optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

class aitlas.models.shallow.**ShallowCNNNetMultilabel**(*config*)

Bases: *BaseMultilabelClassifier*

Simlpe shallow multi-label CNN network for testing purposes

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.starrnn module

StarRNN model for multiclass classification

Note: Adapted from https://github.com/dl4sits/BreizhCrops Original implementation of StarRNN model: https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/models/StarRNN.py Author: Türkoglu Mehmet Özgür <ozgur.turkoglu@geod.baug.ethz.ch>

---

class aitlas.models.starrnn.**StarRNN**(*config*)

Bases: *BaseMulticlassClassifier*

StarRNN model implementation

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *StarRNNSchema*

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

training: bool

class aitlas.models.starrnn.**StarCell**(*input_size*, *hidden_size*, *bias=True*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

reset_parameters()

forward(*x*, *hidden*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.starrnn.**StarLayer**(*input_dim*, *hidden_dim*, *bias=True*, *droput_factor=0.2*, *batch_norm=True*, *layer_norm=False*, *device=device(type='cpu')*)

Bases: `Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

aitlas.models.swin_transformer module

Swin Transformer V2 model for multi-class and multi-label classification tasks.

class aitlas.models.swin_transformer.**SwinTransformer**(*config*)

Bases: *BaseMulticlassClassifier*

A Swin Transformer V2 implementation for multi-class classification tasks.

Note: Based on <https://pytorch.org/vision/stable/models/generated/torchvision.models.swin_v2_s.html#torchvision.models.swin_v2_s>

Initialize a SwinTransformer object with the given configuration.

Parameters

config (*Config schema object*) – A configuration containing model-related settings.

name = 'SwinTransformerV2'

freeze()

Freeze all the layers in the model except for the head. This prevents the gradient computation for the frozen layers during backpropagation.

forward(*x*)

Perform a forward pass through the model.

Parameters
x (*torch.Tensor*) – Input tensor with shape (batch_size, channels, height, width).

Returns
Output tensor with shape (batch_size, num_classes).

Return type
torch.Tensor

training: bool

class aitlas.models.swin_transformer.**SwinTransformerMultilabel**(*config*)

Bases: *BaseMultilabelClassifier*

A Swin Transformer V2 implementation for multi-label classification tasks.

Note: Based on <https://pytorch.org/vision/stable/models/generated/torchvision.models.swin_v2_s.html#torchvision.models.swin_v2_s>

Initialize a SwinTransformerMultilabel object with the given configuration.

Parameters
config (*Config schema object*) – A configuration object containing model-related settings.

name = 'SwinTransformerV2'

freeze()

Freeze all the layers in the model except for the head. This prevents the gradient computation for the frozen layers during backpropagation.

forward(*x*)

Perform a forward pass through the model.

Parameters
x (*torch.Tensor*) – Input tensor with shape (batch_size, channels, height, width).

Returns
Output tensor with shape (batch_size, num_classes).

Return type
torch.Tensor

training: bool

aitlas.models.tempcnn module

Temporal Convolutional Neural Network (TempCNN) model

Note: Adapted from: https://github.com/dl4sits/BreizhCrops

Original implementation(s) of TempCNN model: https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/models/LongShortTermMemory.py and https://github.com/charlotte-pel/temporalCNN

class aitlas.models.tempcnn.**TempCNN**(*config*)

Bases: *BaseMulticlassClassifier*

TempCNN model implementation

Note: Based on <https://github.com/dl4sits/BreizhCrops>

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *TempCNNSchema*

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

training: bool

class aitlas.models.tempcnn.**Conv1D_BatchNorm_Relu_Dropout**(*input_dim*, *hidden_dims*, *kernel_size=5*, *drop_probability=0.5*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*X*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

class aitlas.models.tempcnn.**FC_BatchNorm_Relu_Dropout**(*input_dim*, *hidden_dims*, *drop_probability=0.5*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*X*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: bool

class aitlas.models.tempcnn.**Flatten**

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

aitlas.models.transformer module

Transformer model

Note: Adapted from: https://github.com/dl4sits/BreizhCrops

Original implementation of Transformer model: https://github.com/dl4sits/BreizhCrops/blob/master/breizhcrops/models/TransformerModel.py

class aitlas.models.transformer.**TransformerModel**(*config*)

Bases: *BaseMulticlassClassifier*

Transformer model for multi-class classification model implementation

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *TransformerModelSchema*

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

training: bool

class aitlas.models.transformer.**Flatten**

Bases: Module

Flatten module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*input*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

aitlas.models.unet module

UNet model for segmentation

class aitlas.models.unet.**Unet**(*config*)

Bases: *BaseSegmentationClassifier*

UNet segmentation model implementation.

Note: Based on <https://github.com/qubvel/segmentation_models.pytorch>

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.unet_efficientnet module

aitlas.models.unet_efficientnet.**post_process**(*prediction_directory*, *prediction_csv*)

aitlas.models.unet_efficientnet.**post_process_single**(*sourcefile*, *watershed_line=True*, *conn=2*, *polygon_buffer=0.5*, *tolerance=0.5*, *seed_msk_th=0.75*, *area_th_for_seed=110*, *prediction_threshold=0.5*, *area_th=80*, *contact_weight=1.0*, *edge_weight=0.0*, *seed_contact_weight=1.0*, *seed_edge_weight=1.0*)

aitlas.models.unet_efficientnet.**evaluation**(*prediction_csv*, *gt_csv*)

class aitlas.models.unet_efficientnet.**FocalLoss2d**(*gamma=3*, *ignore_index=255*, *eps=1e-06*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*outputs*, *targets*, *weights=1.0*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.unet_efficientnet.**DiceLoss**(*weight=None*, *per_image=False*, *eps=1e-06*)

Bases: `Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*outputs*, *targets*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the `Module` instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.models.unet_efficientnet.**GenEfficientNet**(*block_args*, *num_classes=1000*, *in_channels=3*, *num_features=1280*, *stem_size=32*, *fix_stem=False*, *channel_multiplier=1.0*, *channel_divisor=8*, *channel_min=None*, *pad_type=''*, *act_layer=<class 'torch.nn.modules.activation.ReLU'>*, *drop_connect_rate=0.0*, *se_kwargs=None*, *norm_layer=<class 'torch.nn.modules.batchnorm.BatchNorm2d'>*, *norm_kwargs=None*, *weight_init='goog'*)

Bases: `Module`

Initializes internal Module state, shared by both nn.Module and ScriptModule.

training: bool

class aitlas.models.unet_efficientnet.**UNetEfficientNet**(*config*)

Bases: *BaseSegmentationClassifier*

Unet EfficientNet model implementation. .. note:: Based on <https://github.com/SpaceNetChallenge/SpaceNet_SAR_Buildings_Solutions/blob/master/1-zbigniewwojna/main.py#L178>

:param config : the configuration for this model :type config : UNetEfficientNetModelSchema

schema

alias of *UNetEfficientNetModelSchema*

forward(*x*, *strip*, *direction*, *coord*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

load_optimizer()

Load the optimizer

load_lr_scheduler()

Load the learning rate scheduler

train_and_evaluate_model(*train_dataset*, *epochs=100*, *model_directory=None*, *save_epochs=10*, *iterations_log=100*, *resume_model=None*, *val_dataset=None*, *run_id=None*, ***kwargs*)

Overridden method for training on the SpaceNet6 data set.

Parameters

- **train_dataset** (SpaceNet6Dataset) –
- **epochs** (*int*) –
- **model_directory** (*str | None*) –
- **save_epochs** (*int*) –
- **iterations_log** (*int*) –
- **resume_model** (*str | None*) –
- **val_dataset** (SpaceNet6Dataset *| None*) –
- **run_id** (*str | None*) –

evaluate(*dataset=None*, *model_path=None*)

Evaluate a model stored in a specified path against a given dataset

Parameters

- **dataset** (SpaceNet6Dataset *| None*) – the dataset to evaluate against
- **model_path** (*str | None*) – the path to the model on disk

Returns

load_model(*file_path*, *optimizer=None*)

Loads a model from a checkpoint

training: bool

aitlas.models.unsupervised module

DeepCluster model

class aitlas.models.unsupervised.**UnsupervisedDeepMulticlassClassifier**(*config*)

Bases: *BaseMulticlassClassifier*

Unsupervised Deep Learning model implementation

Note: Based on Deep Clustering: <https://github.com/facebookresearch/deepcluster>

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

schema

alias of *UnsupervisedDeepMulticlassClassifierSchema*

train_epoch(*epoch*, *dataloader*, *optimizer*, *criterion*, *iterations_log*)

Overriding train epoch to implement the custom logic for the unsupervised classifier

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

aitlas.models.unsupervised.**compute_features**(*dataloader*, *model*, *N*, *batch*, *device*)

Compute features for images

class aitlas.models.unsupervised.**VGG**(*features*, *num_classes*, *sobel*)

Bases: Module

Initializes internal Module state, shared by both nn.Module and ScriptModule.

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

aitlas.models.unsupervised.**make_layers**(*input_dim*, *batch_norm*)

aitlas.models.unsupervised.**vgg16**(*sobel=False*, *bn=True*, *out=1000*)

class aitlas.models.unsupervised.**UnifLabelSampler**(*N*, *images_lists*)

Bases: Sampler

Samples elements uniformely accross pseudolabels.

Parameters

- **N** (*int*) – size of returned iterator.
- **images_lists** – lists of images for each pseudolabel.

generate_indexes_epoch()

aitlas.models.vgg module

VGG16 model

class aitlas.models.vgg.**VGG16**(*config*)

Bases: *BaseMulticlassClassifier*

VGG16 model implementation

Note: Based on: <https://pytorch.org/vision/stable/models/generated/torchvision.models.vgg16.html#torchvision.models.vgg16>

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'VGG16'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

freeze()

extract_features()

Remove final layers if we only need to extract features

training: bool

class aitlas.models.vgg.**VGG19**(*config*)

Bases: *BaseMulticlassClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'VGG19'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.vgg.**VGG16MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'VGG16'

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

extract_features()

Remove final layers if we only need to extract features

freeze()

training: bool

class aitlas.models.vgg.**VGG19MultiLabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'VGG19'

training: bool

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

freeze()

extract_features()

Remove final layers if we only need to extract features

aitlas.models.vision_transformer module

VisionTransformer model (base_patch16_224)

class aitlas.models.vision_transformer.**VisionTransformer**(*config*)

Bases: *BaseMulticlassClassifier*

VisionTransformer model implementation

Note: Based on: <https://github.com/huggingface/pytorch-image-models/tree/main/timm>

BaseModel constructor

Parameters

config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ViT base_patch16_224'

freeze()

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

training: bool

class aitlas.models.vision_transformer.**VisionTransformerMultilabel**(*config*)

Bases: *BaseMultilabelClassifier*

BaseModel constructor

Parameters
config (Config, *optional*) – Configuration object which specifies the details of the model, defaults to None.

name = 'ViT base_patch16_224'

freeze()

training: bool

forward(*x*)

Abstract method implementing the model. Extending classes should override this method. :return: Instance extending *nn.Module* :rtype: nn.Module

5.5 Tasks module

5.5.1 Tasks module

aitlas.tasks.evaluate module

class aitlas.tasks.evaluate.**EvaluateTask**(*model*, *config*)

Bases: *BaseTask*

Parameters
model (BaseModel) –

schema

alias of *EvaluateTaskSchema*

run()

Evaluate the dataset against a given model

aitlas.tasks.extract_features module

class aitlas.tasks.extract_features.**ExtractFeaturesTask**(*model*, *config*)

Bases: *BaseTask*

Parameters
model (BaseModel) –

schema

alias of *ExtractFeaturesTaskSchema*

run()

Do something awesome here

aitlas.tasks.predict module

class aitlas.tasks.predict.**ImageFolderDataset**(*data_dir*, *labels*, *transforms*, *batch_size*)

Bases: *BaseDataset*

BaseDataset constructor

Parameters

config (Config, *contains information for the batch size, number of workers, list of labels, list of transformations*) – Configuration object which specifies the details of the dataset.

class aitlas.tasks.predict.**PredictTask**(*model*, *config*)

Bases: *BaseTask*

Parameters

model (BaseModel) –

schema

alias of *PredictTaskSchema*

run()

Do something awesome here

export_predictions_to_csv(*file*, *fnames*, *probs*, *labels*)

class aitlas.tasks.predict.**PredictSegmentationTask**(*model*, *config*)

Bases: *BaseTask*

Parameters

model (BaseModel) –

schema

alias of *PredictTaskSchema*

run()

Do something awesome here

class aitlas.tasks.predict.**PredictEOPatchTask**(*model*, *config*)

Bases: *BaseTask*

Parameters

model (BaseModel) –

schema

alias of *PredictTaskSchema*

run()

Do something awesome here

export_predictions_to_csv(*file*, *fnames*, *probs*, *labels*)

aitlas.tasks.prepare module

class aitlas.tasks.prepare.**PrepareTask**(*model*, *config*)

Bases: *BaseTask*

If the prepare part (or a version of it) is extensive, you can run it as a separate task

Parameters

model (BaseModel) –

schema
alias of *PrepareTaskSchema*

run()
Do some offline preparation

aitlas.tasks.schemas module

class aitlas.tasks.schemas.**BaseTaskShema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Schema for configuring a base task.

Parameters

- **log** (*bool, optional*) – Flag indicating whether to turn on logging. Default is True.
- **id** (*str, optional*) – Run name/ID for the task. Default is None.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.tasks.schemas.**SplitSetObjectSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Schema for configuring a split dataset object.

Parameters

- **ratio** (*int*) – Ratio of the dataset to include in the split. This is required.
- **file** (*str*) – File containing the indices for the split. This is required.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.tasks.schemas.**SplitObjectSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.tasks.schemas.**SplitTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Schema for configuring a split task.

Parameters

- **data_dir** (*str*) – Path to the dataset on disk. This is required.
- **csv_file** (*str, optional*) – CSV file on disk containing dataset information. Default is None.
- **split** (SplitObjectSchema, *optional*) – Configuration on how to split the dataset. Default is None.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**TrainTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Schema for configuring a training task.

Parameters

- **dataset_config** (ObjectConfig) – Train dataset type and configuration. This is required.
- **epochs** (*int*) – Number of epochs used in training. This is required.
- **model_directory** (*str*) – Directory of the model output. This is required.
- **save_epochs** (*int, optional*) – Number of training steps between model checkpoints. Default is 100.
- **iterations_log** (*int, optional*) – After how many mini-batches do we want to show something in the log. Default is 200.
- **resume_model** (*str, optional*) – File path to the model to be resumed. Default is None.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**TrainAndEvaluateTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Schema for configuring a task that involves training and evaluation.

Parameters

- **epochs** (*int*) – Number of epochs used in training. This is required.
- **model_directory** (*str*) – Directory of the model output. This is required.
- **save_epochs** (*int, optional*) – Number of training steps between model checkpoints. Default is 100.
- **iterations_log** (*int, optional*) – After how many mini-batches do we want to show something in the log. Default is 200.

- **resume_model** (*str, optional*) – File path to the model to be resumed. Default is None.
- **train_dataset_config** (ObjectConfig) – Train dataset type and configuration. This is required.
- **val_dataset_config** (ObjectConfig) – Validation dataset type and configuration. This is required.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**ParameterSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.tasks.schemas.**OptimizeTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Schema for configuring an optimization task.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –

- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**EvaluateTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Schema for configuring an evaluation task.

Parameters

- **dataset_config** (ObjectConfig) – Dataset type and configuration. This is required.
- **model_path** (*str*) – Path to the model. This is required.
- **metrics** (*List[str], optional*) – Metric classes you want to calculate. Default is an empty list.
- **visualizations** (*List[str], optional*) – Visualization classes you want to show. Default is an empty list.
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**PredictTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Schema for configuring a prediction task.

Parameters

- **data_dir** (*str*) – Directory with the image to perform prediction on. This is required.
- **model_path** (*str*) – Path to the model. This is required.
- **output_dir** (*str, optional*) – Folder path where the plot images with predictions will be stored. Default is '/predictions'.
- **output_file** (*str, optional*) – CSV file path where the predictions will be stored. Default is 'predictions.csv'.
- **dataset_config** (ObjectConfig, *optional*) – Dataset type and configuration. Default is None.
- **batch_size** (*int, optional*) – Batch size. Default is 64.
- **labels** (*List[str], optional*) – Labels needed to tag the predictions. Default is None.
- **transforms** (*List[str], optional*) – Classes to run transformations. Default is a list of common torchvision transformations.
- **output_format** (*str, optional*) – Whether to output the predictions to CSV or plots. Default is 'plot'. Must be one of ['plot', 'csv', 'image'].
- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> Field objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**PrepareTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Parameters

- **only** (*types.StrSequenceOrSet | None*) –

- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**ExtractFeaturesTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Schema for configuring a task to extract features from images.

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]

Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

class aitlas.tasks.schemas.**VisualizeSplitSetObjectSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: `Schema`

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.tasks.schemas.**VisualizeSplitObjectSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: Schema

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

class aitlas.tasks.schemas.**VisualizeTaskSchema**(**, only=None, exclude=(), many=False, context=None, load_only=(), dump_only=(), partial=False, unknown=None*)

Bases: *BaseTaskShema*

Parameters

- **only** (*types.StrSequenceOrSet | None*) –
- **exclude** (*types.StrSequenceOrSet*) –
- **many** (*bool*) –
- **context** (*dict | None*) –
- **load_only** (*types.StrSequenceOrSet*) –
- **dump_only** (*types.StrSequenceOrSet*) –
- **partial** (*bool | types.StrSequenceOrSet*) –
- **unknown** (*str | None*) –

opts: SchemaOpts = <marshmallow.schema.SchemaOpts object>

fields: Dict[str, ma_fields.Field]
Dictionary mapping field_names -> `Field` objects

load_fields: Dict[str, ma_fields.Field]

dump_fields: Dict[str, ma_fields.Field]

aitlas.tasks.split module

class aitlas.tasks.split.**BaseSplitTask**(*model*, *config*)
Bases: *BaseTask*

Base task meant to split dataset

Parameters
model (BaseModel) –

schema
alias of *SplitTaskSchema*

is_multilabel = False

extensions = ['.jpg', '.jpeg', '.png', '.ppm', '.bmp', '.pgm', '.tif', '.tiff', '.webp']

run()
Runs the task.

has_val()

is_split_valid()

split()

save_split(*X*, *y*, *file*)

load_images(*data_dir*, *csv_file*, *extensions=None*)
Attempts to read in VOC format, then in internal format, then in folder per class format

make_splits()

perform_split(*X*, *y*, *test_size*)

class aitlas.tasks.split.**RandomSplitTask**(*model*, *config*)
Bases: *BaseSplitTask*

Randomly split a folder containing images

Parameters
model (BaseModel) –

perform_split(*X*, *y*, *test_size*)
Peform actual split using pytorch random split

class aitlas.tasks.split.**StratifiedSplitTask**(*model*, *config*)
Bases: *BaseSplitTask*

Meant for multilabel stratified slit

Parameters
model (BaseModel) –

perform_split(*X*, *y*, *test_size*)
Perform the actual split using sklearn or skmultilearn

aitlas.tasks.train module

class aitlas.tasks.train.**TrainTask**(*model*, *config*)
Bases: *BaseTask*

Parameters
model (BaseModel) –

schema
alias of *TrainTaskSchema*

run()
Do something awesome here

class aitlas.tasks.train.**TrainAndEvaluateTask**(*model*, *config*)
Bases: *BaseTask*

Parameters
model (BaseModel) –

schema
alias of *TrainAndEvaluateTaskSchema*

run()
Do something awesome here

aitlas.tasks.train.**generate_parameters_for_range**(*method*, *parameter*)

aitlas.tasks.train.**generate_parameters**(*method*, *parameters*)
Generate parameters to search

class aitlas.tasks.train.**OptimizeTask**(*model*, *config*)
Bases: *BaseTask*

Optimize certain parameters for the models

Parameters
model (BaseModel) –

schema
alias of *OptimizeTaskSchema*

run()
Do something awesome here

aitlas.tasks.unsupervised_pre_training module

aitlas.tasks.visualize module

class aitlas.tasks.visualize.**VisualizeTask**(*model*, *config*)
Bases: *BaseTask*

Parameters
model (BaseModel) –

schema
alias of *VisualizeTaskSchema*

get_distribution_for_split(*split*, *split_type*)

get_distribution()

run()
Visualize the distribution of the dataset

5.6 Metrics module

5.6.1 Metrics package

aitlas.metrics.classification module

Metrics for classification tasks.

class aitlas.metrics.classification.**AccuracyScore**(***kwargs*)

Bases: *BaseMetric*

Accuracy score class, inherits from BaseMetric.

name = 'accuracy'

key = 'accuracy'

calculate(*y_true*, *y_pred*)

Computes the Accuracy score.

Given model predictions for a target variable, it calculates the accuracy score as the number of correct predictionsdivided by the total number of predictions.

Parameters

- **y_true** (*array-like of arbitrary size*) – The ground truth values for the target variable.
- **y_pred** (*array-like of identical size as y_true*) – The prediction values for the target variable.

Returns
A number in [0, 1] where, 1 is a perfect classification.

Return type
float

class aitlas.metrics.classification.**AveragedScore**(***kwargs*)

Bases: *BaseMetric*

Average score class. Inherits from BaseMetric.

calculate(*y_true*, *y_pred*)

It calculates the score for each class and then averages the results. The type of average is {'micro', 'macro', 'weighted'}:

***'micro': Calculate metrics globally by counting the total true positives,**
false negatives and false positives.

***'macro': Calculate metrics for each label, and find their unweighted**
mean. This does not take label imbalance into account.

***'weighted': Calculate metrics for each label, and find their average, weighted**
by support (the number of true instances for each label). This alters 'macro' to account for label imbalance.

Parameters

- **y_true** (*array-like*) – The ground truth labels
- **y_pred** (*array-like*) – The predicted labels

Returns
A dictionary with the micro, macro and weighted average scores

Return type
dict

Raises
ValueError – If the shapes of y_pred and y_true do not match.

class aitlas.metrics.classification.**PrecisionScore**(***kwargs*)
Bases: *AveragedScore*

Precision score class, inherits from AveragedScore.

name = 'precision'

key = 'precision'

class aitlas.metrics.classification.**RecallScore**(***kwargs*)
Bases: *AveragedScore*

Precision score class, inherits from AveragedScore.

name = 'recall'

key = 'recall'

class aitlas.metrics.classification.**F1Score**(***kwargs*)
Bases: *AveragedScore*

name = 'f1 score'

key = 'f1_score'

aitlas.metrics.segmentation module

Metrics for segmentation tasks.

class aitlas.metrics.segmentation.**F1ScoreSample**(***kwargs*)
Bases: *BaseMetric*

Calculates the F1 score metric for binary segmentation tasks.

name = 'F1 Score'

key = 'f1_score'

calculate(*y_true*, *y_pred*, *beta=1*, *eps=1e-07*)
Calculate the F1 Score.

Parameters
- **y_true** (*list or numpy array*) – True labels
- **y_pred** (*list or numpy array*) – Predicted labels
- **beta** (*float*) – Weight of precision in the combined score. Default is 1.
- **eps** (*float*) – Small value to prevent zero division. Default is 1e-7.

Returns
F1 score

Return type
float

Raises
ValueError – If the shapes of y_pred and y_true do not match.

class aitlas.metrics.segmentation.**IoU**(***kwargs*)

Bases: *BaseMetric*

Calculates the Intersection over Union (IoU) metric for binary segmentation tasks.

name = 'IoU'

key = 'iou'

calculate(*y_true*, *y_pred*, *eps=1e-07*)

Calculate the IoU score.

Parameters

- **y_true** (*list or numpy array*) – True labels
- **y_pred** (*list or numpy array*) – Predicted labels
- **eps** (*float*) – Small value to prevent zero division. Default is 1e-7.

Returns
IoU score

Return type
float

Raises
ValueError – If the shapes of y_pred and y_true do not match.

class aitlas.metrics.segmentation.**Accuracy**(***kwargs*)

Bases: *BaseMetric*

Calculates the accuracy metric.

name = 'Accuracy'

key = 'accuracy'

calculate(*y_true*, *y_pred*)

Calculate accuracy.

Parameters

- **y_true** (*list or numpy array*) – True labels
- **y_pred** (*list or numpy array*) – Predicted labels

Returns
Accuracy score

Return type
float

class aitlas.metrics.segmentation.**DiceCoefficient**(***kwargs*)

Bases: *BaseMetric*

A Dice Coefficient metic, used to evaluate the similarity of two sets.

Note: More information on its Wikipedia page: https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient

---

name = 'DiceCoefficient'

key = 'dice_coefficient'

calculate(*y_true*, *y_pred*)

Method to compute the Dice coefficient.

Given two sets X and Y, the coefficient is calculated as:

$$DSC = 2 * |XintersectionY|/|X| + |Y|, where|X|and|Y|arethecardinalitiesofthetwosets.$$

Note: Based on the implementation at: https://github.com/CosmiQ/cresi/blob/master/cresi/net/pytorch_utils/loss.py#L47

Parameters

- **y_true** (*list or numpy array*) – The ground truth values for the target variable. Can be array-like of arbitrary size.
- **y_pred** (*list or numpy array*) – The prediction values for the target variable. Must be of identical size as y_true.

Returns
A number in [0, 1] where 0 equals no similarity and 1 is maximum similarity.

Return type
float

Raises
ValueError – If the shapes of y_pred and y_true do not match.

class aitlas.metrics.segmentation.**FocalLoss**(*alpha=1*, *gamma=2*, *logits=True*, *reduce=True*, ***kwargs*)

Bases: *BaseMetric*

Class for calculating Focal Loss, a loss metric that extends the binary cross entropy loss. Focal loss reduces the relative loss for well-classified examples and puts more focus on hard, misclassified examples. Computed as: .. math:: alpha * (1-bce_loss)**gamma

---

Note: For more information, refer to the papers: https://paperswithcode.com/method/focal-loss, and: https://amaarora.github.io/2020/06/29/FocalLoss.html

---

Intilisation.

Parameters

- **alpha** (*int*) – Weight parameter. Default is 1.
- **gamma** (*int*) – Focusing parameter. Default is 2.
- **logits** (*bool*) – Controls whether probabilities or raw logits are passed. Default is True.
- **reduce** (*bool*) – Specifies whether to reduce the loss to a single value. Default is True.
- **kwargs** – Any key word arguments to be passed to the base class

name = 'FocalLoss'

key = 'focal_loss'

calculate(*y_true*, *y_pred*)

Method to compute the focal loss.

Note: Based on the implementation at: https://www.kaggle.com/c/tgs

---

Parameters

y_true – The ground truth values for the target variable. Can be array-like of arbitrary size. :type y_true: list or numpy array :param y_pred: The prediction values for the target variable. Must be of identical size as y_true. :type y_pred: list or numpy array :return: The focal loss between y_pred and y_true. :rtype: float :raises ValueError: If the shapes of y_pred and y_true do not match.

class aitlas.metrics.segmentation.**CompositeMetric**(*metrics=None*, *weights=None*, ***kwargs*)

Bases: *BaseMetric*

A class for combining multiple metrics.

Initialisation.

Parameters

- **metrics** (*list*) – A list of metrics that subclass the BaseMetric class and have valid implementation of calculate(y_true, y_pred). Default is None.
- **weights** (*list*) – A list of floats who sum up to 1. Default is None.
- **kwargs** – Any key word arguments to be passed to the base class

Raises

ValueError – If the length of metrics and weights is not equal or if the sum of weights is not equal to one.

name = 'CompositeMetric'

key = 'composite_metric'

calculate(*y_true*, *y_pred*)

Method to calculate the weighted sum of the metric values.

Parameters

- **y_true** (*list or numpy array*) – The ground truth values for the target variable. Can be array-like of arbitrary size.
- **y_pred** (*list or numpy array*) – The prediction values for the target variable. Must be of identical size as y_true.

Returns

The weighted sum of each metric value.

Return type

float

Raises

ValueError – If the shapes of y_pred and y_true do not match.

5.7 Visualizations module

5.7.1 Visualizations package

aitlas.visualizations.classification module

Classes and methods for visualizations for classification tasks.

aitlas.visualizations.classification.**plot_confusion_matrix**(*confusion_matrix*, *axes*, *class_label*, *class_names*, *fontsize=14*)

Plots a confusion matrix.

Parameters

- **confusion_matrix** (*array-like of shape (n_classes, n_classes)*) – The confusion matrix to plot.
- **axes** (*matplotlib.axes.Axes*) – The matplotlib axes object to plot on.
- **class_label** (*str*) – The label of the class for the confusion matrix.
- **class_names** (*list of str*) – The names of the classes.
- **fontsize** (*int, optional*) – The fontsize for the plot, defaults to 14.

aitlas.visualizations.classification.**plot_multilabel_confusion_matrix**(*cm_array*, *labels*, *dataset_name*, *output_file*)

Plots multiple confusion matrices in a .pdf format for multilabel tasks.

Parameters

- **cm_array** (*list of array-like of shape (n_classes, n_classes)*) – The array of confusion matrices.
- **labels** (*list of str*) – The labels for the classes.
- **dataset_name** (*str*) – The name of the dataset.
- **output_file** (*str*) – The file path to save the plot.

aitlas.visualizations.classification.**plot_multiclass_confusion_matrix**(*cm_array*, *labels*, *dataset_name*, *output_file*)

Plots multiple confusion matrices .pdf format useful for multiclass tasks.

Parameters

- **cm_array** (*list of array-like of shape (n_classes, n_classes)*) – The array of confusion matrices.
- **labels** (*list of str*) – The labels for the classes.
- **dataset_name** (*str*) – The name of the dataset.
- **output_file** (*str*) – The file path to save the plot.

class aitlas.visualizations.classification.**PrecisionRecallCurve**(*y_true*, *y_pred*, *y_prob*, *labels*, *file*, ***kwargs*)

Bases: *BaseDetailedVisualization*

plot()

Generates and plots the precision recall curve.

Returns
matplotlib.figure.Figure object with the plot

Return type
matplotlib.figure.Figure

class aitlas.visualizations.classification.**ImageLabelsVisualization**(*y_true*, *y_pred*, *y_prob*, *labels*, *file*, ***kwargs*)

Bases: *BaseDetailedVisualization*

Class for visualising predictions for an image.

Initialize the ImageLabelsVisualization class.

Parameters

- **y_true** (*array-like of shape (n_samples,)*) – Ground truth (correct) labels.
- **y_pred** (*array-like of shape (n_samples,)*) – Predicted labels, as returned by a classifier.
- **y_prob** (*list of float*) – The predicted probabilities.
- **labels** (*list of str*) – The labels for the classes.
- **file** (*str*) – The file path to save the plot.
- **kwargs** – Additional keyword arguments.

plot()

Plots the image with the predictions.

Returns
matplotlib.figure.Figure object with the plot

Return type
matplotlib.figure.Figure

plot_prediction(*img*, *probs*, *classes*)

Display image and predictions from model

Parameters

- **img** (*array-like or PIL image*) – Image to plot.
- **prob** (*list of float*) – The predicted probabilities.
- **classes** (*list of str*) – The labels for the classes.

Returns
matplotlib.figure.Figure object with the plot

Return type
matplotlib.figure.Figure

aitlas.visualizations.classification.**display_image_labels**(*image*, *y_true*, *y_pred*, *y_prob*, *labels*, *output_file*)

aitlas.visualizations.classification.**precision_recall_curve**(*y_true*, *y_pred*, *y_prob*, *labels*, *output_file*)

aitlas.visualizations.eopatch module

Method for visualising predictions in EOpatch format for multi-temporal data. Useful for croptype classification tasks.

aitlas.visualizations.eopatch.**display_eopatch_predictions**(*eopatches_path*, *patch*, *y_pred*, *test_index*, *y_true*, *classmapping*)

Displays the predictions of an EOPatch.

Parameters

- **eopatches_path** (*str*) – The path to the directory containing EOPatches.
- **patch** (*str*) – The specific patch to be displayed.
- **y_pred** (*array-like of shape (n_samples,)*) – The predicted labels, as returned by a classifier.
- **test_index** (*pandas.DataFrame*) – The indices of the test set.
- **y_true** (*array-like of shape (n_samples,)*) – Ground truth (correct) labels.
- **classmapping** (*pandas.DataFrame*) – A mapping from class labels to class names.

Returns
matplotlib.figure.Figure object with the plot

Return type
matplotlib.figure.Figure

aitlas.visualizations.grad_cam module

Classes and methods for GRAD-CAM visualizations, used for classification tasks.

Note: Based on the implementation at: https://github.com/jacobgil/pytorch-grad-cam

class aitlas.visualizations.grad_cam.**ActivationsAndGradients**(*model*, *target_layers*, *reshape_transform*)

Bases: object

Class for extracting activations and registering gradients from targetted intermediate layers.

Parameters

- **model** – The model to be evaluated
- **target_layers** – The target layers from which to extract activations and gradients

- **reshape_transform** – A function to reshape the activation and gradient data

save_activation(*module*, *input*, *output*)

Saves an activation.

Parameters

- **module** – The module from which to save the activation
- **input** – The input data to the module
- **output** – The output data from the module

save_gradient(*module*, *input*, *output*)

Saves the gradient.

Parameters

- **module** – The module from which to save the gradient
- **input** – The input data to the module
- **output** – The output data from the module

release()

aitlas.visualizations.grad_cam.**get_2d_projection**(*activation_batch*)

aitlas.visualizations.grad_cam.**scale_cam_image**(*cam*, *target_size=None*)

aitlas.visualizations.grad_cam.**show_cam_on_image**(*img*, *mask*, *use_rgb=False*, *colormap=2*, *image_weight=0.5*)

This function overlays the cam mask on the image as an heatmap.

By default the heatmap is in BGR format. :param img: The base image in RGB or BGR format. :param mask: The cam mask. :param use_rgb: Whether to use an RGB or BGR heatmap, this should be set to True if 'img' is in RGB format. :param colormap: The OpenCV colormap to be used. :param image_weight: The final result is image_weight * img + (1-image_weight) * mask. :returns: The default image with the cam overlay.

Parameters

- **img** (*ndarray*) –
- **mask** (*ndarray*) –
- **use_rgb** (*bool*) –
- **colormap** (*int*) –
- **image_weight** (*float*) –

Return type
ndarray

class aitlas.visualizations.grad_cam.**ClassifierOutputTarget**(*category*)

Bases: object

aitlas.visualizations.grad_cam.**reshape_transform**(*tensor*, *height=14*, *width=14*)

class aitlas.visualizations.grad_cam.**BaseCAM**(*model*, *target_layers*, *use_cuda=False*, *reshape_transform=None*, *compute_input_gradient=False*, *uses_gradients=True*)

Bases: object

Parameters

- **model** (*Module*) –
- **target_layers** (*List[Module]*) –
- **use_cuda** (*bool*) –
- **reshape_transform** (*Callable*) –
- **compute_input_gradient** (*bool*) –
- **uses_gradients** (*bool*) –

get_cam_weights(*input_tensor*, *target_layers*, *targets*, *activations*, *grads*)

Parameters

- **input_tensor** (*Tensor*) –
- **target_layers** (*List[Module]*) –
- **targets** (*List[Module]*) –
- **activations** (*Tensor*) –
- **grads** (*Tensor*) –

Return type
ndarray

get_cam_image(*input_tensor*, *target_layer*, *targets*, *activations*, *grads*, *eigen_smooth=False*)

Parameters

- **input_tensor** (*Tensor*) –
- **target_layer** (*Module*) –
- **targets** (*List[Module]*) –
- **activations** (*Tensor*) –
- **grads** (*Tensor*) –
- **eigen_smooth** (*bool*) –

Return type
ndarray

forward(*input_tensor*, *targets*, *eigen_smooth=False*)

Parameters

- **input_tensor** (*Tensor*) –
- **targets** (*List[Module]*) –
- **eigen_smooth** (*bool*) –

Return type
ndarray

get_target_width_height(*input_tensor*)

Parameters
input_tensor (*Tensor*) –

Return type
Tuple[int, int]

compute_cam_per_layer(*input_tensor*, *targets*, *eigen_smooth*)

Parameters

- **input_tensor** (*Tensor*) –
- **targets** (*List[Module]*) –
- **eigen_smooth** (*bool*) –

Return type
ndarray

aggregate_multi_layers(*cam_per_target_layer*)

Parameters
cam_per_target_layer (*ndarray*) –

Return type
ndarray

forward_augmentation_smoothing(*input_tensor*, *targets*, *eigen_smooth=False*)

Parameters

- **input_tensor** (*Tensor*) –
- **targets** (*List[Module]*) –
- **eigen_smooth** (*bool*) –

Return type
ndarray

class aitlas.visualizations.grad_cam.**GradCAM**(*model*, *target_layers*, *use_cuda=False*, *reshape_transform=None*)

Bases: *BaseCAM*

Gradient-weighted Class Activation Mapping (Grad-CAM) class.

Parameters

- **model** – The model to be evaluated
- **target_layers** – The target layers from which to extract activations and gradients
- **use_cuda** – Whether to use CUDA for computation (default: False)
- **reshape_transform** – A function to reshape the activation and gradient data (default: None)

get_cam_weights(*input_tensor*, *target_layer*, *target_category*, *activations*, *grads*)

aitlas.visualizations.segmentation module

Classes and methods for visualizations for segmentation tasks.

class aitlas.visualizations.segmentation.**ImageMaskPredictionVisualization**(*y_true*, *y_pred*, *y_prob*, *labels*, *file*, ***kwargs*)

Bases: *BaseDetailedVisualization*

Class for visualizing the image mask predictions.

Initialisation

Parameters

- **y_true** (*array-like of shape (n_samples,)*) – The ground truth labels
- **y_pred** (*array-like of shape (n_samples,)*) – The predicted labels
- **y_prob** (*list of float*) – The predicted probabilities
- **labels** (*list of str*) – The class labels
- **file** (*str*) – The output file path

plot()

Plots the image mask predictions and saves the plot to the output file.

plot_segmenation(*img*, *probs*, *labels*)

Displays the image and the predicted segmentation masks for each label.

Parameters

- **img** (*array-like or PIL image*) – The input image
- **probs** (*list of float*) – The predicted probabilities
- **labels** (*list of str*) – The class labels

Returns
The figure containing the plots

Return type
matplotlib.figure.Figure

aitlas.visualizations.segmentation.**display_image_segmentation**(*image*, *y_true*, *y_pred*, *y_prob*, *labels*, *file*)

Displays the predicted segmentation masks for each label.

Parameters

- **image** (*array-like or PIL image*) – The input image
- **y_true** (*array-like of shape (n_samples,)*) – The ground truth labels
- **y_pred** (*array-like of shape (n_samples,)*) – The predicted labels
- **y_prob** (*list of float*) – The predicted probabilities
- **labels** (*list of str*) – The class labels
- **file** (*str*) – The output file path

aitlas.visualizations.segmentation.**save_predicted_masks**(*y_pred*, *labels*, *base_filepath_name*)

Saves the predicted masks to the specified file path.

Parameters

- **y_pred** (*array-like of shape (n_samples,)*) – The predicted labels
- **labels** (*list of str*) – The class labels
- **base_filepath_name** (*str*) – The base file path name

5.8 Utils module

5.8.1 Utils module

aitlas.utils.segmentation_losses module

Loss functions for image segmentation

class aitlas.utils.segmentation_losses.**DiceLoss**

Bases: Module

Dice Loss for image segmentation. Expects sigmoided inputs and binary targets. ..note:: Implementation from: kaggle.com/bigironsphere/loss-function-library-keras-pytorch

forward(*inputs*, *targets*, *smooth=1*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

class aitlas.utils.segmentation_losses.**FocalLoss**

Bases: Module

Focal Loss for image segmentation. Expects sigmoided inputs and binary targets. ..note:: Implementation from: kaggle.com/bigironsphere/loss-function-library-keras-pytorch

ALPHA = 0.8

GAMMA = 2

forward(*inputs*, *targets*, *alpha=0.8*, *gamma=2*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

---

training: bool

aitlas.utils.utils module

aitlas.utils.utils.**get_class**(*class_name*)

Returns the class type for a given class name. Expects a string of type *module.submodule.Class*

aitlas.utils.utils.**current_ts**()

Returns current timestamp in secs

aitlas.utils.utils.**pil_loader**(*file*, *convert_to_grayscale=False*)

Opens an image from disk

aitlas.utils.utils.**tiff_loader**(*file*)

Opens a tiff image from disk

aitlas.utils.utils.**image_loader**(*file_path*, *convert_to_grayscale=False*)

Opens an image from disk

Parameters

- **file_path** (*str*) – path to the image
- **convert_to_grayscale** (*bool*) – whether to convert the image to grayscale

aitlas.utils.utils.**image_invert**(*file_path*, *convert_to_grayscale=False*)

Inverts an image from disk

Parameters

- **file_path** (*str*) – path to the image
- **convert_to_grayscale** (*bool*) – whether to convert the image to grayscale

aitlas.utils.utils.**stringify**(*obj*)

Stringify whatever object you have

aitlas.utils.utils.**parse_img_id**(*file_path*, *orients*)

Parses direction, strip and coordinate components from a SpaceNet6 image filepath.

aitlas.utils.utils.**split_images**(*images_dir*, *ext_images*, *masks_dir*, *ext_masks*, *output_dir*, *target_size*)

aitlas.utils.utils.**load_voc_format_dataset**(*dir_path*, *csv_file_path*)

Loads a dataset in the Pascal VOC format. It expects a *multilabels.txt* file and *images* in the root folder

aitlas.utils.utils.**has_file_allowed_extension**(*file_path*, *extensions*)

Checks if a file is an allowed extension.

Pararm file_path
path to a file

Parameters
extensions (*list*) – list of allowed extensions

Returns
True if the file is an allowed extension, False otherwise

Return type
bool

aitlas.utils.utils.**load_folder_per_class_dataset**(*dir*, *extensions=None*)

aitlas.utils.utils.**load_aitlas_format_dataset**(*file_path*)

Reads the images from a CSV. Format: (image_path, class_name)

aitlas.utils.utils.**submit_inria_results**(*input_dir*, *output_dir*)

aitlas.utils.utils.**save_best_model**(*model*, *model_directory*, *epoch*, *optimizer*, *loss*, *start*, *run_id*)

Saves the model on disk :param model: model to save :type model: torch.nn.Module :param model_directory: directory where to save the model :type model_directory: str :param epoch: current epoch :type epoch: int :param optimizer: optimizer used for training :type optimizer: torch.optim.Optimizer :param loss: loss value :type loss: float :param start: start time :type start: float :param run_id: run id :type run_id: str

aitlas.utils.utils.**collate_fn**(*batch*)

5.9 Clutering module

5.9.1 Clustering module

aitlas.clustering.kmeans module

class aitlas.clustering.kmeans.**Kmeans**(*k*)

Bases: object

cluster(*data*, *verbose=False*)

Performs k-means clustering.

Parameters

x_data (*np.array (N * dim)*) – data to cluster

aitlas.clustering.pic module

class aitlas.clustering.pic.**PIC**(*args=None*, *sigma=0.2*, *nnn=5*, *alpha=0.001*, *distribute_singletons=True*)

Bases: object

Class to perform Power Iteration Clustering on a graph of nearest neighbors. Arguments for consistency with k-means init:

Parameters

- **sigma** (*float*) – bandwith of the Gaussian kernel (default 0.2)
- **nnn** (*int*) – number of nearest neighbors (default 5)
- **alpha** (*float*) – parameter in PIC (default 0.001)
- **distribute_singletons** (*bool*) – If True, reassign each singleton to the cluster of its closest nonsingleton nearest neighbors (up to nnn nearest neighbors).
- **images_lists** (*list of lists of ints*) – for each cluster, the list of image indexes belonging to this cluster

cluster(*data*, *verbose=False*)

aitlas.clustering.utils module

aitlas.clustering.utils.**preprocess_features**(*npdata*, *pca=256*)

Preprocess an array of features.

Parameters

- **npdata** (*np.array (N * dim)*) – features to preprocess
- **pca** (*int*) – dim of output

Returns

data PCA-reduced, whitened and L2-normalized

Return type

np.array (N * pca)

aitlas.clustering.utils.**make_graph**(*xb*, *nnn*)

Builds a graph of nearest neighbors.

Parameters

- **xb** (*np.array (N * dim)*) – data
- **nnn** (*int*) – number of nearest neighbors

Returns
list for each data the list of ids to its nnn nearest neighbors

Returns
list for each data the list of distances to its nnn NN

Return type
np.array (N * nnn)

class aitlas.clustering.utils.**ReassignedDataset**(*image_indexes*, *pseudolabels*, *dataset*)

Bases: Dataset

A dataset where the new images labels are given in argument.

Parameters

- **image_indexes** (*list of ints*) – list of data indexes
- **pseudolabels** (*list of ints*) – list of labels for each data
- **dataset** (*list of tuples with paths to images*) – initial dataset
- **transform** (*callable, optional*) – a function/transform that takes in an PIL image and returns a transformed version

make_dataset(*image_indexes*, *pseudolabels*)

aitlas.clustering.utils.**cluster_assign**(*images_lists*, *dataset*)

Creates a dataset from clustering, with clusters as labels.

Params images_lists
for each cluster, the list of image indexes belonging to this cluster

Params dataset
initial dataset

Returns
dataset with clusters as labels

Return type
ReassignedDataset(torch.utils.data.Dataset)

aitlas.clustering.utils.**run_kmeans**(*x*, *nmb_clusters*, *verbose=False*)

Runs kmeans on 1 GPU. :param x: data :type x: np.array (N * dim) :param nmb_clusters: number of clusters :type nmb_clusters: int :return: list of ids for each data to its nearest cluster :rtype: list of ints

aitlas.clustering.utils.**arrange_clustering**(*images_lists*)

aitlas.clustering.utils.**make_adjacencyW**(*I*, *D*, *sigma*)

Create adjacency matrix with a Gaussian kernel.

Parameters

- **I** (*numpy array*) – for each vertex the ids to its nnn linked vertices + first column of identity.
- **D** (*numpy array*) – for each data the l2 distances to its nnn linked vertices + first column of zeros.

- **sigma** (*float*) – bandwith of the Gaussian kernel.

Returns
affinity matrix of the graph.

Return type
scipy.sparse.csr_matrix

aitlas.clustering.utils.**run_pic**(*I*, *D*, *sigma*, *alpha*)

Run PIC algorithm

aitlas.clustering.utils.**find_maxima_cluster**(*W*, *v*)

5.10 Run module

Run an AiTALS workflow with a configuration file.

aitlas.run.**run**(*rank*, *config*)

Load model, if specified in the configuration file, and run the task. It initializes the model and the task, and then runs the task.

aitlas.run.**main**(*config_file*)

This is the main entry function for the toolbox It specifies the configuration file, loads the configuration, and runs the task.

Python Module Index

a

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W