Beetles

Release 1.5

KP Labs

Mar 26, 2021

CONTENTS:

CHAPTER ONE

INSTALLATION

For the environment management in our repo, we utilize *conda*.

Required dependencies are stored in the *environment.yml* file. To create the environment with required packages, execute the following command:

```
conda env create -f environment.yml
```

Because of the fact that we utilize the Xillinx DNNDK tool for model quantization and compilation, the *tensorflow* package has to be installed manually. Please refer to the official documentation, where the process is fully described: (https://www.xilinx.com/support/documentation/sw_manuals/ai_inference/v1_6/ug1327-dnndk-user-guide.pdf - Chapter 1: Quick Start - Tensorflow Version: Installing with Anaconda).

Keep in mind, that the DNNDK tool requires **Ubuntu** 14.04, 16.04 or 18.04.

CHAPTER

TWO

MACHINE-LEARNING

2.1 cloud_detection package

2.1.1 Subpackages

2.1.2 Submodules

2.1.3 cloud_detection.data_gen module

Generator based data loader for Cloud38 and L8CCA clouds segmentation datasets.

class `cloud_detection.data_gen.`**`DG_38Cloud`**(*files*, *batch_size*, *balance_classes=False*, *balance_snow=False*, *dim=(384, 384)*, *shuffle=True*, *with_gt=True*)

Bases: `tensorflow.python.keras.utils.data_utils.Sequence`

Data generator for Cloud38 clouds segmentation dataset. Works with Keras generators.

Prepare generator and init paths to files containing image channels.

Parameters

- **`files`** (`List[Dict[str, Path]]`) – List of dicts containing paths to rgb channels of each image in dataset.
- **`batch_size`** (`int`) – size of generated batches, only one batch is loaded to memory at a time.
- **`balance_classes`** (`bool`) – if True balance classes.
- **`balance_snow`** (`bool`) – if True balances patches with snow and clouds.
- **`dim`** (`Tuple[int, int]`) – Tuple with x, y image dimensions.
- **`shuffle`** (`bool`) – if True shuffles dataset on each epoch end.
- **`with_gt`** (`bool`) – if True returns y along with x.

`on_epoch_end`()

Triggered after each epoch, if shuffle is True randomises file indexing.

class `cloud_detection.data_gen.`**`DG_L8CCA`**(*img_path*, *img_name*, *batch_size*, *patch_size=384*, *shuffle=True*)

Bases: `tensorflow.python.keras.utils.data_utils.Sequence`

Data generator for Cloud38 clouds segmentation dataset. Works with Keras generators.

Prepare generator and init paths to files containing image channels.

Parameters

- **`img_path`** (`Path`) – path to the main directory with test scenes.
- **`img_name`** (`str`) – name of the image to generate patches from.
- **`batch_size`** (`int`) – size of generated batches, only one batch is loaded to memory at a time.
- **`patch_size`** (`int`) – size of the patches.
- **`shuffle`** (`bool`) – if True shuffles dataset on each epoch end.

`on_epoch_end``()`
Triggered after each epoch, if shuffle is True randomises file indexing.

`cloud_detection.data_gen.`**`load_image_paths`**`(`*base_path*, *patches_path=None*, *split_ratios=[1.0]*, *shuffle=True*, *img_id=None*`)`
Build paths to all files containing image channels.

Parameters

- **`base_path`** (`Path`) – root path containing directories with image channels.
- **`patches_path`** (`Optional[Path]`) – path to images patches names to load (if None, all patches will be used).
- **`split_ratios`** (`List[float]`) – list containing split ratios, splits should add up to one.
- **`shuffle`** (`bool`) – whether to shuffle image paths.
- **`img_id`** (`Optional[str]`) – image ID; if specified, load paths for this image only.

Return type `List[List[Dict[str, Path]]]`

Returns list with paths to image files, separated into splits. Structured as: list_of_splits[list_of_files['file_channel', Path]]

`cloud_detection.data_gen.`**`main`**`()`
Demo data loading and present one data sample.

`cloud_detection.data_gen.`**`strip_nir`**`(`*hyper_img*`)`
Strips nir channel so image can be displayed.

Parameters **`hyper_img`** (`ndarray`) – image with shape (x, y, 4) where fourth channel is nir.

Return type `ndarray`

Returns image with shape (x, y, 3) with standard RGB channels.

2.1.4 cloud_detection.evaluate_38Cloud module

Get evaluation metrics for given model on 38-Cloud test set.

`cloud_detection.evaluate_38Cloud.`**`evaluate_model`**`(`*model*, *thr*, *dpath*, *gtpath*, *vpath*, *rpath*, *vids*, *batch_size*, *img_ids=None*, *mlflow=False*, *run_name=None*`)`
Get evaluation metrics for given model on 38-Cloud testset.

Parameters

- **`model`** (`Model`) – trained model to make predictions.
- **`thr`** (`float`) – threshold to be used during evaluation.

- **dpath** (Path) – path to dataset.
- **gtpath** (Path) – path to dataset ground truths.
- **vpath** (Path) – path to dataset (false color) visualisation images.
- **rpath** (Path) – path to direcotry where results and artifacts should be logged.
- **vids** (Tuple[str]) – tuple of ids of images which should be used to create visualisations. If contains '*' visualisations will be created for all images in the dataset.
- **batch_size** (int) – size of generated batches, only one batch is loaded to memory at a time.
- **img_ids** (Optional[List[str]]) – if given, process only these images.
- **mlflow** – whether to use MLFlow.
- **run_name** – name of the run.

Return type Tuple

Returns evaluation metrics.

cloud_detection.evaluate_38Cloud.**get_full_scene_img**(*path*, *img_id*)

Get image of given id as np.array with values in range 0 to 1.

Parameters

- **path** (Path) – path to the dataset.
- **img_id** (str) – ID of the image.

Return type ndarray

Returns image.

cloud_detection.evaluate_38Cloud.**get_img_pred**(*path*, *img_id*, *model*, *batch_size*, *patch_size=384*)

Generates prediction for a given image.

Parameters

- **path** (Path) – path containing directories with image channels.
- **img_id** (str) – ID of the considered image.
- **model** (Model) – trained model to make predictions.
- **batch_size** (int) – size of generated batches, only one batch is loaded to memory at a time.
- **patch_size** (int) – size of the image patches.

Return type Tuple[ndarray, float]

Returns prediction for the given image along with evaluation time.

cloud_detection.evaluate_38Cloud.**get_img_pred_shape**(*files*, *patch_size*)

Infers shape of the predictions of the considered image.

Parameters

- **files** (List[Dict[str, Path]]) – paths to patch files; structured as: list_of_files['file_channel', Path].
- **patch_size** (int) – size of the image patches.

Return type Tuple

Returns shape of the predictions of the considered image.

`cloud_detection.evaluate_38Cloud.`**`load_img_gt`**(*path*, *fname*)

Load image ground truth.

Parameters

- **`path`** (`Path`) – path containing image gts.
- **`fname`** (`str`) – image gt file name.

Return type `ndarray`

Returns image ground truth.

2.1.5 cloud_detection.evaluate_L8CCA module

Get evaluation metrics for given model on L8CCA dataset.

`cloud_detection.evaluate_L8CCA.`**`build_rgb_scene_img`**(*path*, *img_id*)

Build displayable rgb image out channel slices.

Parameters

- **`path`** (`Path`) – path to directory with images.
- **`img_id`** (`str`) – id of image to be loaded.

Return type `ndarray`

Returns rgb image in numpy array.

`cloud_detection.evaluate_L8CCA.`**`evaluate_model`**(*model*, *thr*, *dpath*, *rpath*, *vids*, *batch_size*, *img_ids=None*, *mlflow=False*, *run_name=None*)

Get evaluation metrics for given model on 38-Cloud test set.

Parameters

- **`model`** (`Model`) – trained model to make predictions.
- **`thr`** (`float`) – threshold.
- **`dpath`** (`Path`) – path to dataset.
- **`rpath`** (`Path`) – path to direcotry where results and artifacts should be logged.
- **`vids`** (`Tuple[str]`) – tuple of ids of images which should be used to create visualisations. If contains '*' visualisations will be created for all images in the dataset.
- **`batch_size`** (`int`) – size of generated batches, only one batch is loaded to memory at a time.
- **`img_ids`** (`Optional[List[str]]`) – if given, process only these images.
- **`mlflow`** – whether to use MLFlow.
- **`run_name`** – name of the run.

Return type `Tuple`

Returns evaluation metrics.

`cloud_detection.evaluate_L8CCA.`**`get_img_pred`**(*path*, *img_id*, *model*, *batch_size*, *patch_size=384*)

Generates prediction for a given image.

Parameters

- **`path`** (`Path`) – path containing directories with image channels.
- **`img_id`** (`str`) – ID of the considered image.
- **`model`** (`Model`) – trained model to make predictions.
- **`batch_size`** (`int`) – size of generated batches, only one batch is loaded to memory at a time.
- **`patch_size`** (`int`) – size of the image patches.

Return type `Tuple[ndarray, float]`

Returns prediction for a given image along with evaluation time.

`cloud_detection.evaluate_L8CCA.`**`load_img_gt`**`(`*path*, *fname*`)`

Load image ground truth.

Parameters

- **`path`** (`Path`) – path containing image gts.
- **`fname`** (`str`) – image gt file name.

Return type `ndarray`

Returns image ground truth.

2.1.6 cloud_detection.losses module

Custom loss functions and metrics for segmentation.

`class` `cloud_detection.losses.`**`DiceCoefMetric`**`(`*smooth=1e-07*`)`

Bases: `object`

Dice coefficient metric for segmentation like tasks. Internally uses Jaccard index metric.

Create dice coef metric callable class.

Default smoothness coefficient comes from Cloud-Net example. :type smooth: `float` :param smooth: Small smoothing value to prevent zero division.

`__call__``(`*y_true*, *y_pred*`)`

Calcualate dice coef metric.

Parameters

- **`y_true`** (`ndarray`) – True labels.
- **`y_pred`** (`ndarray`) – Predicted labels.

Returns Dice coef metric score value.

`class` `cloud_detection.losses.`**`JaccardIndexLoss`**`(`*smooth=1e-07*`)`

Bases: `object`

Jaccard index loss for segmentation like tasks.

Create Jaccard index loss callable class.

Default smoothness coefficient comes from Cloud-Net example. :type smooth: `float` :param smooth: Small smoothing value to prevent zero division.

`__call__``(`*y_true*, *y_pred*`)`

Calculate Jaccard index loss.

Parameters

- **y_true** (ndarray) – True labels.
- **y_pred** (ndarray) – Predicted labels.

Return type float

Returns Jaccard index loss score value

class cloud_detection.losses.**JaccardIndexMetric**(*smooth=1e-07*)

Bases: object

Jaccard index metric for segmentation like tasks. Contrary to the JaccardIndexLoss this operates on classes/categories not on probabilities.

Create Jaccard index metric loss callable class. Default smoothness coefficient comes from Cloud-Net example.

Parameters **smooth** (float) – Small smoothing value to prevent zero division.

__call__(*y_true*, *y_pred*)

Calculate Jaccard index metric.

Parameters

- **y_true** (ndarray) – True labels.
- **y_pred** (ndarray) – Predicted labels.

Return type float

Returns Jaccard index metric score value.

cloud_detection.losses.**f1_score**(*y_true*, *y_pred*)

Calculate f1 score.

Parameters

- **y_true** (ndarray) – True labels.
- **y_pred** (ndarray) – Predicted labels.

Return type float

Returns F1 score.

cloud_detection.losses.**precision**(*y_true*, *y_pred*)

Calculate precision score.

Parameters

- **y_true** (ndarray) – True labels.
- **y_pred** (ndarray) – Predicted labels.

Return type float

Returns Precision score.

cloud_detection.losses.**recall**(*y_true*, *y_pred*)

Calculate recall score.

Parameters

- **y_true** (ndarray) – True labels.
- **y_pred** (ndarray) – Predicted labels.

Return type float

Returns Recall score.

`cloud_detection.losses.`**`specificity`**(*y_true*, *y_pred*)

Calculate specificity score.

Parameters

- **`y_true`** (`ndarray`) – True labels.
- **`y_pred`** (`ndarray`) – Predicted labels.

Return type `float`

Returns Specificity score.

2.1.7 cloud_detection.main module

Train and test the models for cloud detection.

`cloud_detection.main.`**`main`**(*run_name*, *train_path*, *C38_path*, *C38_gtpath*, *L8CCA_path*, *vpath*, *rpath*, *ppath*, *vids*, *mlflow*, *train_size*, *train_img*, *balance_train_dataset*, *balance_val_dataset*, *balance_snow*, *snow_imgs_38Cloud*, *snow_imgs_L8CCA*, *batch_size*, *thr*, *learning_rate*, *bn_momentum*, *epochs*, *stopping_patience*)

Train and test the U-Net model using 38-Cloud and L8CCA datasets.

Parameters

- **`run_name`** (`str`) – name of the run.
- **`train_path`** (`str`) – path to train dataset.
- **`C38_path`** (`str`) – path to 38-Cloud dataset.
- **`C38_gtpath`** (`str`) – path to 38-Cloud groundtruth.
- **`L8CCA_path`** (`str`) – path to L8CCA dataset.
- **`vpath`** (`str`) – path to 38-Cloud dataset (false color) visualisation images.
- **`rpath`** (`str`) – path to directory where results and artifacts should be logged (randomly named directory will be created to store the results).
- **`ppath`** (`str`) – path to file with names of training patches (if None, all training patches will be used).
- **`vids`** (`Tuple[str]`) – tuple of ids of images which should be used to create visualisations. If contains '*' visualisations will be created for all images in the datasets.
- **`mlflow`** (`bool`) – whether to use mlflow
- **`train_size`** (`float`) – proportion of the training set (the rest goes to validation set).
- **`train_img`** (`str`) – image ID for training; if specified, load training patches for this image only.
- **`balance_train_dataset`** (`bool`) – whether to balance train dataset.
- **`balance_val_dataset`** (`bool`) – whether to balance val dataset.
- **`balance_snow`** (`bool`) – whether to balance snow images.
- **`snow_imgs_38Cloud`** (`List[str]`) – list of 38-Cloud snow images IDs for testing.
- **`snow_imgs_L8CCA`** (`List[str]`) – list of L8CCA snow images IDs for testing.

- **batch_size** (int) – size of generated batches, only one batch is loaded to memory at a time.
- **thr** (float) – threshold for determining whether pixels contain the clouds (if None, threshold will be determined automatically).
- **learning_rate** (float) – learning rate for training.
- **bn_momentum** (float) – momentum of the batch normalization layer.
- **epochs** (int) – number of epochs.
- **stopping_patience** (int) – patience param for early stopping.

2.1.8 cloud_detection.models module

Keras models for cloud detection.

cloud_detection.models.**unet**(*input_size*, *bn_momentum*)

Simple U-Net model based on model from https://medium.com/analytics-vidhya/creating-a-very-simple-u-net-model- with-pytorch-for-semantic-segmentation-of-satellite-images-223aa216e705 consisting of 3 contract blocks and 3 expand blocks.

Parameters

- **input_size** (int) – Number of input channels, i.e., the number of spectral bands.
- **bn_momentum** (float) – Momentum of the batch normalization layer.

Return type Model

Returns U-Net model.

2.1.9 cloud_detection.train_model module

Train the models for cloud detection.

cloud_detection.train_model.**train_model**(*dpath*, *rpath*, *ppath*, *train_size*, *batch_size*, *balance_train_dataset*, *balance_val_dataset*, *balance_snow*, *train_img*, *bn_momentum*, *learning_rate*, *stopping_patience*, *epochs*, *mlflow*)

Train the U-Net model using 38-Cloud dataset.

Parameters

- **dpath** (Path) – path to dataset.
- **rpath** (Path) – path to direcotry where results and artifacts should be logged.
- **ppath** (Path) – path to file with names of training patches (if None, all training patches will be used).
- **train_size** (float) – proportion of the training set (the rest goes to validation set).
- **batch_size** (int) – size of generated batches, only one batch is loaded to memory at a time.
- **balance_train_dataset** (bool) – whether to balance train dataset.
- **balance_val_dataset** (bool) – whether to balance val dataset.
- **balance_snow** (bool) – whether to balance snow.

- **train_img** (str) – image ID for training; if specified, load training patches for this image only.
- **bn_momentum** (float) – momentum of the batch normalization layer.
- **learning_rate** (float) – learning rate for training.
- **stopping_patience** (int) – patience param for early stopping.
- **epochs** (int) – number of epochs.
- **mlflow** (bool) – whether to use mlflow

Return type Model

Returns trained model.

2.1.10 cloud_detection.utils module

Various utilities, running tools, img editing etc.

class cloud_detection.utils.**MLFlowCallback**

Bases: tensorflow.python.keras.callbacks.Callback

on_epoch_end(*epoch*, *logs=None*)

Triggered after each epoch, logging metrics to MLFlow.

Parameters

- **epoch** (int) – index of epoch.
- **logs** (Optional[Dict]) – logs for MLFlow.

cloud_detection.utils.**false_negatives**(*y_true*, *y_pred*)

Calculate matrices indicating false negatives in given predictions.

Parameters

- **y_true** (ndarray) – True labels.
- **y_pred** (ndarray) – Predicted labels.

Returns Array with values indicating false negatives in predictions.

cloud_detection.utils.**false_positives**(*y_true*, *y_pred*)

Calculate matrices indicating false positives in given predictions.

Parameters

- **y_true** (ndarray) – True labels.
- **y_pred** (ndarray) – Predicted labels.

Returns Array with values indicating false positives in predictions.

cloud_detection.utils.**get_metrics**(*gt*, *pred*, *metric_fns*)

Calculates evaluation metrics for a given image predictions.

Parameters

- **gt** (ndarray) – image ground truth.
- **pred** (ndarray) – image predictions.
- **metric_fns** (List[Callable]) – list of metric functions.

Return type Dict

Returns evaluation metrics.

`cloud_detection.utils.`**`make_paths`**`(`**args*`)`

Make Paths out of strings.

Params strings to make into Paths.

Return type `Tuple[Path]`

Returns Paths made out of input strings.

`cloud_detection.utils.`**`open_as_array`**`(`*channel_files*`)`

Load image as array from given files. Normalises images on load.

Parameters **`channel_files`** (`Dict[str, Path]`) – Dict with paths to files containing each channel of an image, keyed as 'red', 'green', 'blue', 'nir'.

Return type `ndarray`

Returns given image as a single numpy array.

`cloud_detection.utils.`**`overlay_mask`**`(`*image*, *mask*, *rgb_color*, *overlay_intensity=0.5*`)`

Overlay a mask on image for visualization purposes.

Parameters

- **`image`** (`ndarray`) – Image on which mask should be overlaid.
- **`mask`** (`ndarray`) – Mask which should be overlaid on the image.
- **`rgb_color`** (`Tuple[float, float, float]`) – Tuple of three floats containing intensity of RGB channels of created mask. RBG values can be in range 0 to 1 or 0 to 255 depending on the image and mask values range. This will effectively set color of the overlay mask.
- **`overlay_intensity`** (`float`) – Intensity of the overlaid mask. Should be between 0 and 1.

Return type `ndarray`

Returns mask overlaid on image.

`cloud_detection.utils.`**`pad`**`(`*img*, *patch_size=384*`)`

Padding of an image to divide it into patches.

Parameters

- **`img`** (`ndarray`) – image to pad.
- **`patch_size`** (`int`) – size of the patches.

Return type `ndarray`

Returns padded image.

`cloud_detection.utils.`**`save_vis`**`(`*img_id*, *img_vis*, *img_pred*, *img_gt*, *rpath*`)`

Save visualisations set for img of given id. Visualisations set includes: * Mask overlay of uncertain regions of segmentation. * Ground truth mask. * Prediction mask. * TP, FP, FN mask overlays.

Parameters

- **`img_id`** (`str`) – Id of visualised img, will be used for naming saved artifacts.
- **`img_vis`** (`ndarray`) – RGB image.
- **`img_pred`** (`ndarray`) – Prediction mask, result of segmentation.
- **`img_gt`** (`ndarray`) – Ground truth mask.

- **rpath** (`Path`) – Path where artifacts should be saved.

`cloud_detection.utils.`**`setup_mlflow`**`(`*run_name*`)`

Start mlflow run with given name.

Parameters **`run_name`** (`str`) – name of the run.

`cloud_detection.utils.`**`true_positives`**`(`*y_true*, *y_pred*`)`

Calculate matrices indicating true positives in given predictions.

Parameters

- **`y_true`** (`ndarray`) – True labels.
- **`y_pred`** (`ndarray`) – Predicted labels.

Return type `ndarray`

Returns Array with values indicating true positives in predictions.

`cloud_detection.utils.`**`unpad`**`(`*img*, *gt_shape*`)`

Unpadding of an image to return it to its original shape.

Parameters

- **`img`** (`ndarray`) – image to unpad.
- **`gt_shape`** (`Tuple`) – shape of the original image.

Return type `ndarray`

Returns unpadded image.

2.1.11 cloud_detection.validate module

Perform various validation tasks, like making ROC, precision-recall, thresholding etc.

`cloud_detection.validate.`**`datagen_to_gt_array`**`(`*datagen*`)`

Load all gt masks from data generator and store them in RAM at once inside a np.array.

Parameters **`datagen`** (`Sequence`) – data generator to load gt masks from.

Return type `ndarray`

Returns array of gt masks.

`cloud_detection.validate.`**`find_best_thr`**`(`*fpr*, *tpr*, *thr*`)`

Find best threshold based on ROC curve.

Parameters

- **`fpr`** (`ndarray`) – False positives rate ROC axis.
- **`tpr`** (`ndarray`) – True positives rate ROC axis.
- **`thr`** (`ndarray`) – Thresholds corresponding to ROC points.

Return type `float`

Returns Best thr based on minimal distance to point (0, 1) on ROC.

`cloud_detection.validate.`**`find_nearest`**`(`*array*, *value*`)`

In an array find a value that is closest to the given one.

Parameters

- **`array`** (`Sequence`) – Sequence inside which closest value will be found.

- **`value`** (`float`) – Value for which the closest one in the array will be found.

Return type `float`

Returns value inside the 'array' param that is closest to the 'value' param.

`cloud_detection.validate.`**`make_activation_hist`**(*y_pred*, *output_dir*)

Make histogram of prediction activations.

Parameters

- **`y_pred`** (`ndarray`) – Array of predictions.
- **`output_dir`** (`Path`) – Path to directory where hist should be saved.

`cloud_detection.validate.`**`make_precision_recall`**(*y_gt*, *y_pred*, *output_dir*, *thr_marker=None*)

Make precision recall curve and save it.

Parameters

- **`y_gt`** (`ndarray`) – Array of ground truth masks.
- **`y_pred`** (`ndarray`) – Array of predictions.
- **`output_dir`** (`Path`) – Path to directory where curve should be saved.
- **`thr_marker`** (`Optional[float]`) – Extra threshold level to mark on the curve.

`cloud_detection.validate.`**`make_roc`**(*y_gt*, *y_pred*, *output_dir*, *thr_marker=None*)

Make ROC curve and save it, get best thr based on the ROC.

Parameters

- **`y_gt`** (`ndarray`) – Array of ground truth masks.
- **`y_pred`** (`ndarray`) – Array of predictions.
- **`output_dir`** (`Path`) – Path to directory where ROC should be saved.
- **`thr_marker`** (`Optional[float]`) – extra threshold level to mark on ROC.

Return type `float`

Returns Best thr based on minimal distance to point (0, 1) on ROC.

`cloud_detection.validate.`**`make_validation_insights`**(*model*, *datagen*, *output_dir*)

Make validation insights including:

- Making ROC and finding best thr on ROC.
- Making precision-recall curve.

Parameters

- **`model`** (`Model`) – Model to validate.
- **`datagen`** (`Sequence`) – Validation data generator.
- **`output_dir`** (`Path`) – Path to directory where artifacts should be saved.

Return type `float`

Returns Best thr based on ROC.

`cloud_detection.validate.`**`validate_demo`**()

Demo of the validation function.

2.1.12 Module contents

2.2 ml_intuition package

2.2.1 Subpackages

ml_intuition.data package

Submodules

ml_intuition.data.input_fn module

File containing functions which calibrate input for frozen graphs used for quantization in scripts/quantize.sh. All arguments to such functions are passed through the shell environment, because they are executed through the internal decent tools.

`ml_intuition.data.input_fn.`**`calibrate_2d_input`**`(`*iter*`)`
Return dictionary with a batch of the training data to be used for quantization calibration. One dimension and the end is added and the min max normalization is performed.

Parameters **`iter`** (`int`) – Int object indicating the calibration step

Return type `Dict[str, ndarray]`

Returns Dict with name of the input node as key and training samples in np.ndarray as value

ml_intuition.data.io module

All I/O related functions

`ml_intuition.data.io.`**`extract_set`**`(`*data_path*, *dataset_key*`)`
Function for loading a h5 format dataset as a dictionary of samples, labels, min and max values.

Parameters

- **`data_path`** (`str`) – Path to the dataset.
- **`dataset_key`** (`str`) – Key for dataset.

Return type `Dict[str, Union[ndarray, float]]`

Returns Dictionary containing labels, data, min and max values.

`ml_intuition.data.io.`**`load_metrics`**`(`*experiments_path*, *filename=None*`)`
Load metrics to a dictionary.

Parameters

- **`experiments_path`** (`str`) – Path to the experiments directory.
- **`filename`** (`Optional[str]`) – Name of the file holding metrics. Defaults to 'inference_metrics.csv'.

Return type `Dict[List, List]`

Returns Dictionary containing all metric names and values from all experiments.

`ml_intuition.data.io.`**`load_npy`**`(`*data_file_path*, *gt_input_path*, *use_unmixing=False*`)`
Load .npy data and GT from specified paths

Parameters

- **`data_file_path`** (`str`) – Path to the data .npy file
- **`gt_input_path`** (`str`) – Path to the GT .npy file
- **`use_unmixing`** (`bool`) – Boolean indicating whether to perform experiments on the unmixing datasets, where classes in each pixel are present as abundance fractions.

Return type `Tuple[ndarray, ndarray]`

Returns Tuple with loaded data and GT

`ml_intuition.data.io.`**`load_pb`**(*path_to_pb*)
Load .pb file as a graph

Parameters **`path_to_pb`** (`str`) – Path to the .pb file

Return type `GraphDef`

Returns Loaded graph

`ml_intuition.data.io.`**`load_processed_h5`**(*data_file_path*)
Load processed dataset containing the train, validation and test subsets with corresponding samples and labels.

Parameters **`data_file_path`** (`str`) – Path to the .h5 file.

Return type `Dict`

Returns Dictionary containing train, validation and test subsets.

`ml_intuition.data.io.`**`load_satellite_h5`**(*data_file_path*)
Load hyperspectral cube and ground truth transformation matrix from .h5 file

Parameters **`data_file_path`** (`str`) – Path to the .h5 file

Return type `Tuple[ndarray, ndarray]`

Returns Hyperspectral cube and transformation matrix, both as np.ndarray

`ml_intuition.data.io.`**`load_tiff`**(*file_path*)
Load tiff image into np.ndarray

Parameters **`file_path`** (`str`) – Path to the .tiff file

Return type `ndarray`

Returns Loaded image as np.ndarray

`ml_intuition.data.io.`**`read_min_max`**(*path*)
Read min and max value from a .csv file

Parameters **`path`** (`str`) – Path to the .csv file containing min and max value

Return type `Tuple[float, float]`

Returns Tuple with min and max

`ml_intuition.data.io.`**`save_confusion_matrix`**(*matrix*, *dest_path*, *filename='confusion_matrix'*)
Save confusion matrix to provided destination. If filename is not provided ‘confusion_matrix.csv’ will be used.

Parameters

- **`matrix`** (`ndarray`) – Matrix to save
- **`dest_path`** (`str`) – Destination folder
- **`filename`** (`str`) – Name of the file, defaults to ‘confusion_matrix.csv’

Return type `None`

Returns None

`ml_intuition.data.io.`**`save_md5`**`(`*output_path*, *train_x*, *train_y*, *val_x*, *val_y*, *test_x*, *test_y*`)`

Save provided data as .md5 file

Parameters

- **`output_path`** (`str`) – Path to the filename
- **`train_x`** (`ndarray`) – Train set
- **`train_y`** (`ndarray`) – Train labels
- **`val_x`** (`ndarray`) – Validation set
- **`val_y`** (`ndarray`) – Validation labels
- **`test_x`** (`ndarray`) – Test set
- **`test_y`** (`ndarray`) – Test labels

Return type `None`

Returns None

`ml_intuition.data.io.`**`save_metrics`**`(`*dest_path*, *metrics*, *file_name=None*`)`

Save given dictionary of metrics.

Parameters

- **`dest_path`** (`str`) – Destination path.
- **`file_name`** (`Optional[str]`) – Name to save the file.
- **`metrics`** (`Dict[str, List]`) – Dictionary containing all metrics.

`ml_intuition.data.io.`**`save_ml_report`**`(`*output_dir_path*, *model_name*, *test_report*, *best_params*, *train_fraction*`)`

Save the test report and parameters obtained from the best model.

Parameters

- **`output_dir_path`** (`str`) – Path to the output directory.
- **`model_name`** (`str`) – Name of the model.
- **`test_report`** (`DataFrame`) – Report over the test set.
- **`best_params`** (`Dict`) – Best parameters found by the grid search.
- **`norm`** – Normalization used for the dataset.
- **`train_fraction`** (`float`) – Fraction of training samples.

Return type `None`

Returns None.

ml_intuition.data.loggers module

Helper function logging data into MLFlow

`ml_intuition.data.loggers.`**`log_dict_as_str_to_mlflow`**(*dict_as_string*)

Log a string which represents a dictionary to MLflow

Parameters **`dict_as_string`** (`str`) – A string with dictionary format

Return type `None`

Returns None

`ml_intuition.data.loggers.`**`log_metrics_to_mlflow`**(*metrics*, *fair=False*)

Log provided metrics to mlflow

Parameters

- **`metrics`** (`Dict[str, float]`) – Metrics in a dictionary
- **`fair`** (`bool`) – Whether to add '_fair' suffix to the metrics name

Return type `None`

Returns None

`ml_intuition.data.loggers.`**`log_params_to_mlflow`**(*args*)

Log provided arguments as dictionary to mlflow.

Parameters **`args`** (`Dict`) – Arguments to log

Return type `None`

`ml_intuition.data.loggers.`**`log_tags_to_mlflow`**(*run_name*)

Log tags to mlflow based on provided args

Parameters **`run_name`** (`str`) – Name of the current run

Return type `None`

Returns None

ml_intuition.data.noise module

`class` `ml_intuition.data.noise.`**`BaseNoise`**(*params*)

Bases: `abc.ABC`

`abstract __call__`(**args*, ***kwargs*)

Each subclass should implement this method.

Parameters

- **`args`** – Arbitrary list of arguments.
- **`kwargs`** – Arbitrary dictionary of arguments.

`static get_proba`(*n_samples*, *prob*)

Return type `int`

`class` `ml_intuition.data.noise.`**`Gaussian`**(*params*)

Bases: *`ml_intuition.data.noise.BaseNoise`*

`__call__`(*data*, *labels*)

Perform Gaussian noise injection.

Parameters

- **`data`** (`ndarray`) – Input data that will undergo noise injection.
- **`labels`** (`ndarray`) – Class value for each data point.

Return type `List[ndarray]`

Returns List containing the noisy data and the class labels.

class `ml_intuition.data.noise.`**`Impulsive`**(*params*)

Bases: *`ml_intuition.data.noise.BaseNoise`*

`__call__`(*data*, *labels*)

Perform impulsive noise injection.

Parameters

- **`data`** (`ndarray`) – Input data that will undergo noise injection.
- **`labels`** (`ndarray`) – Class value for each data point.

Return type `List[ndarray]`

Returns List containing the noisy data and the class labels.

class `ml_intuition.data.noise.`**`Params`**(*pa: float = None*, *pb: float = None*, *bc: bool = None*, *mean: float = None*, *std: float = None*, *pw: float = None*)

Bases: `tuple`

Parameters of noise injection.

pa - Fraction of noisy pixels, the number of affected samples is calculated by: floor(n_samples * pa).

pb - Fraction of noisy bands. When established the number of samples that undergo noise injection, for each sample the: floor(n_bands * pb) bands are affected.

bc - Boolean indicating whether the indexes of affected bands are constant for each sample. When set to: False, different bands can be augmented with noise for each pixel.

mean - Gaussian noise parameter, the mean of the normal distribution.

std - Standard deviation of the normal distribution for Gaussian noise.

pw - Ratio of whitened pixels for the affected set of samples in the Impulsive noise injection.

Create new instance of Params(pa, pb, bc, mean, std, pw)

property bc

Alias for field number 2

property mean

Alias for field number 3

property pa

Alias for field number 0

property pb

Alias for field number 1

property pw

Alias for field number 5

property std

Alias for field number 4

class `ml_intuition.data.noise.`**`Shot`**(*params*)

Bases: *`ml_intuition.data.noise.BaseNoise`*

`__call__`(*data*, *labels*)

Perform shot noise injection.

Parameters

- **`data`** (`ndarray`) – Input data that will undergo noise injection.
- **`labels`** (`ndarray`) – Class value for each data point.

Return type `List[ndarray]`

Returns List containing the noisy data and the class labels.

`ml_intuition.data.noise.`**`get_all_noise_functions`**(*noise*)

Get all specified noise functions.

Parameters **`noise`** (`List[str]`) – List the of noise injection methods.

Return type `List`

Returns List of all noise injection functions.

`ml_intuition.data.noise.`**`get_noise_functions`**(*noise*, *noise_params*)

Helper function for getting all noise injector instances.

Parameters

- **`noise`** (`List[str]`) – List the of noise injection methods.
- **`noise_params`** (`str`) – Parameters of the noise injections.

Return type `List[`*`BaseNoise`*`]`

Returns List of instances of noise injectors functions.

`ml_intuition.data.noise.`**`inject_noise`**(*data_source*, *affected_subsets*, *noise_injectors*, *noise_params*)

Inject noise into given subsets.

Parameters

- **`data_source`** (`Dict`) – Dictionary containing subsets.
- **`affected_subsets`** (`List[str]`) – List of names of subsets that will undergo noise injection.
- **`noise_injectors`** (`List[str]`) – List of names of noise injection methods.
- **`noise_params`** (`str`) – Parameters of the noise injection.

ml_intuition.data.preprocessing module

All function related to the data preprocessing step of the pipeline.

`ml_intuition.data.preprocessing.`**`align_ground_truth`**(*cube_2d_shape*, *ground_truth*, *cube_to_gt_transform*)

Align original labels to match the satellite hyperspectral cube using transformation matrix

Parameters

- **`cube_2d_shape`** (`Tuple[int, int]`) – Shape of the hyperspectral data cube
- **`ground_truth`** (`ndarray`) – Original labels as 2D array

- **`cube_to_gt_transform`** (`ndarray`) – Cube to ground truth transformation matrix

Return type `ndarray`

Returns Transformed ground truth

`ml_intuition.data.preprocessing.`**`get_padded_cube`**(*data*, *padding_size*)
Pad the cube with zeros

Parameters

- **`data`** (`ndarray`) – Data to pad
- **`padding_size`** (`int`) – Size of the padding for each side

Return type `ndarray`

Returns Padded cube

`ml_intuition.data.preprocessing.`**`normalize_labels`**(*labels*)
Normalize labels so that they always start from 0

Parameters **`labels`** (`ndarray`) – labels to normalize

Return type `ndarray`

Returns Normalized labels

`ml_intuition.data.preprocessing.`**`remove_nan_samples`**(*data*, *labels*)
Remove samples which contain only nan values

Parameters

- **`data`** (`ndarray`) – Data with dimensions [SAMPLES, . . .]
- **`labels`** (`ndarray`) – Corresponding labels

Return type `Tuple`[`ndarray`, `ndarray`]

Returns Data and labels with removed samples containing nans

`ml_intuition.data.preprocessing.`**`reshape_cube_to_2d_samples`**(*data*, *labels*, *channels_idx=0*, *use_unmixing=False*)

Reshape the data and labels from [CHANNELS, HEIGHT, WIDTH] to [PIXEL,CHANNELS, 1], so it fits the 2D Convolutional modules.

Parameters

- **`data`** (`ndarray`) – Data to reshape.
- **`labels`** (`ndarray`) – Corresponding labels.
- **`channels_idx`** (`int`) – Index at which the channels are located in the provided data file.
- **`use_unmixing`** (`bool`) – Boolean indicating whether to perform experiments on the unmixing datasets, where classes in each pixel are present as fractions.

Returns Reshape data and labels

Return type tuple with reshaped data and labels `ml_intuition.data.preprocessing.`**`reshape_cube_to_3d_samples`**(*data*, *labels*, *neighborhood_size=5*, *background_label=0*, *channels_idx=0*, *use_unmixing=False*)

Reshape data to a array of dimensionality: [N_SAMPLES, HEIGHT, WIDTH, CHANNELS] and the labels to dimensionality of: [N_SAMPLES, N_CLASSES]

Parameters

- **`data`** (`ndarray`) – Data passed as array.
- **`labels`** (`ndarray`) – Labels passed as array.
- **`neighborhood_size`** (`int`) – Length of the spatial patch.
- **`background_label`** (`int`) – Label to filter out the background.
- **`channels_idx`** (`int`) – Index of the channels.
- **`use_unmixing`** (`bool`) – Boolean indicating whether to perform experiments on the unmixing datasets, where classes in each pixel are present as abundances fractions.

Return type Tuple of data and labels reshaped to 3D format.

`ml_intuition.data.preprocessing.`**`spectral_angle_mapper`**(*data*, *endmembers*, *channel_idx*)

Calculate the spectral angle mapper between the HSI pixels and endmembers.

Parameters

- **`data`** (`ndarray`) – Data cube.
- **`endmembers`** (`ndarray`) – The matrix containing the spectra for each class of dimensionality [n_classes, n_bands].
- **`channel_idx`** (`int`) – Index of bands in HSI.

Return type `Tuple`[`ndarray`, `ndarray`]

Returns The tuple of both, normalized and unnormalized angle matrix for each pixel in HSI.

`ml_intuition.data.preprocessing.`**`train_val_test_split`**(*data*, *labels*, *train_size=0.8*, *val_size=0.1*, *stratified=True*, *seed=0*)

Split the data into train, val and test sets. The size of the training set is set by the train_size parameter. All the remaining samples will be treated as a test set

Parameters

- **`data`** (`ndarray`) – Data with the [SAMPLES, . . .] dimensions
- **`labels`** (`ndarray`) – Vector with corresponding labels
- **`train_size`** (`Union`[`List`, `float`, `int`]) – If float, should be between 0.0 and 1.0, If stratified = True, it represents percentage of each class to be extracted. If float and stratified = False, it represents percentage of the whole dataset to be extracted with samples drawn randomly, regardless of their class. If int and stratified = True, it represents number of samples to be drawn from each class. If int and stratified = False, it represents overall number of samples to be drawn regardless of their class, randomly. Defaults to 0.8
- **`val_size`** (`float`) – Should be between 0.0 and 1.0. Represents the percentage of each class from the training set to be extracted as a validation set, defaults to 0.1
- **`stratified`** (`bool`) – Indicated whether the extracted training set should be stratified, defaults to True

- **seed** (int) – Seed used for data shuffling

Return type Tuple[ndarray, ndarray, ndarray, ndarray, ndarray, ndarray]

Returns train_x, train_y, val_x, val_y, test_x, test_y

Raises **AssertionError** – When wrong type is passed as train_size

ml_intuition.data.transforms module

Module containing all the transformations that can be done on a dataset.

class ml_intuition.data.transforms.**BaseTransform**

Bases: abc.ABC

abstract __call__(**args*, ***kwargs*)

Each subclass should implement this method.

Parameters

- **args** – Arbitrary list of arguments.
- **kwargs** – Arbitrary dictionary of arguments.

class ml_intuition.data.transforms.**ExtractCentralPixelSpectrumTransform**(*neighborhood_size*)

Bases: *ml_intuition.data.transforms.BaseTransform*

Extract central pixel from each spatial sample.

Parameters **neighborhood_size** (int) – The spatial size of the patch.

__call__(*samples*, *labels*)

" Transform the labels for unsupervised unmixing problem. The label is the central pixel of each sample patch.

Parameters

- **samples** (ndarray) – Input samples.
- **labels** (ndarray) – Central pixel of each sample.

Return type List[ndarray]

Returns List containing the input samples and its targets as central pixel.

class ml_intuition.data.transforms.**MinMaxNormalize**(*min_*, *max_*)

Bases: *ml_intuition.data.transforms.BaseTransform*

Normalize each sample.

Parameters

- **min** – Minimum value of features.
- **max** – Maximum value of features.

__call__(*samples*, *labels*)

" Perform min-max normalization on the passed samples.

Parameters

- **samples** (ndarray) – Input samples that will undergo normalization.
- **labels** (ndarray) – Class values for each sample.

Return type List[ndarray]

Returns List containing the normalized samples and the class labels.

class `ml_intuition.data.transforms.`**`OneHotEncode`**(*n_classes*)

Bases: *`ml_intuition.data.transforms.BaseTransform`*

Initializer of the one-hot encoding transformation.

Parameters **`n_classes`** (`int`) – Number of classes.

`__call__`(*samples*, *labels*)

Perform one-hot encoding on the passed labels.

Parameters

- **`samples`** (`ndarray`) – Input samples.
- **`labels`** (`ndarray`) – Class values for each sample that will undergo one-hot encoding.

Return type `List[ndarray]`

Returns List containing the samples and the one-hot encoded class labels.

class `ml_intuition.data.transforms.`**`PerBandMinMaxNormalization`**(*min_*, *max_*)

Bases: *`ml_intuition.data.transforms.BaseTransform`*

`SPECTRAL_DIM = -1`

`__call__`(*samples*, *labels*)

Perform per-band min-max normalization. Each band is treated as a separate feature.

Parameters

- **`samples`** (`ndarray`) – Input samples that will undergo transformation.
- **`labels`** (`ndarray`) – Abundance vectors for each sample.

Return type `List[ndarray]`

Returns List containing the normalized samples and the abundance vectors.

static **`get_min_max_vectors`**(*data_cube*)

Get the min-max vectors for each spectral band.

Parameters **`data_cube`** (`ndarray`) – Hyperspectral data cube, with bands in the last dimension and spatial features in the first two dimensions.

Return type `Dict[str, ndarray]`

Returns Dictionary containing the min as well as the max vectors for each band.

class `ml_intuition.data.transforms.`**`RNNSpectralInputTransform`**

Bases: *`ml_intuition.data.transforms.BaseTransform`*

`__call__`(*samples*, *labels*)

” Transform the input samples to fit the recurrent neural network (RNN) input. This is performed for the pixel-based model; the input sample includes only spectral bands.

Parameters

- **`samples`** (`ndarray`) – Input samples that will undergo the transformation.
- **`labels`** (`ndarray`) – Class values for each sample.

Return type `List[ndarray]`

Returns List containing the normalized samples and the class labels.

class ml_intuition.data.transforms.**SpectralTransform**(***kwargs*)

Bases: *ml_intuition.data.transforms.BaseTransform*

Initializer of the spectral transformation.

__call__(*samples*, *labels*)

Transform 1D samples along the spectral axis. Only the spectral features are present for each sample in the dataset.

Parameters

- **samples** (ndarray) – Input samples that will undergo transformation.
- **labels** (ndarray) – Class value for each samples.

Return type List[ndarray]

Returns List containing the transformed samples and the class labels.

ml_intuition.data.transforms.**apply_transformations**(*data*, *transformations*)

Apply each transformation from provided list

Parameters

- **data** (Dict) – Dictionary with 'data' and 'labels' keys holding np.ndarrays
- **transformations** (List[*BaseTransform*]) – List of transformations

Return type Dict

Returns Transformed data, in the same format as input

Module contents

2.2.2 Submodules

2.2.3 ml_intuition.models module

All models that are used in the project.

class ml_intuition.models.**Ensemble**(*models=None*, *voting='hard'*)

Bases: object

Ensemble for using multiple models for prediction

Parameters

- **models** (Union[List[Sequential], List[str], Sequential, None]) – Either list of tf.keras.models.Sequential models, or a list of paths to the models (can't mix both).
- **voting** (str) – If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities. If 'classifier', a Random Forest trained on train set predictions is used as a voter.

MODELS = {'RFR': <class 'sklearn.ensemble._forest.RandomForestRegressor'>, 'RFC': <cla

generate_models_with_noise(*copies=5*, *mean=None*, *std=None*, *seed=None*)

Generate new models by injecting Gaussian noise into the original model's weights.

Parameters

- **copies** (int) – Number of models to generate

- **mean** (Optional[float]) – Mean used to draw noise from normal distribution. If None, it will be calculated from the layer itself.
- **std** (Optional[float]) – Standard deviation used to draw noise from normal distribution. If None, it will be calculated from the layer itself.
- **seed** – Seed for random number generator.

Return type None

Returns None

static inject_noise_to_weights(*weights*, *mean*, *std*)
Inject noise into all layers

Parameters

- **weights** (List[ndarray]) – List of weights for each layer
- **mean** (float) – Mean used to draw noise from normal distribution.
- **std** (float) – Std used to draw noise from normal distribution.

Return type List[ndarray]

Returns Modified list of weights

predict(*data*, *batch_size=1024*)
Return predicted classes

Parameters

- **data** (Union[ndarray, List[ndarray]]) – Either a single dataset which will be fed into all of the models, or a list of datasets, unique for each model
- **batch_size** (int) – Size of the batch used for prediction

Return type ndarray

Returns Predicted classes

predict_probabilities(*data*, *batch_size=1024*)
Return predicted classes

Parameters

- **data** (Union[ndarray, List[ndarray]]) – Either a single dataset which will be fed into all of the models, or a list of datasets, unique for each model
- **batch_size** (int) – Size of the batch used for prediction

Return type ndarray

Returns Predicted probabilities for each model and class. Shape: [Models, Samples, Classes]

train_ensemble_predictor(*data*, *labels*, *predictor=None*, *model_params=None*)

vote(*predictions*)
Perform voting process on provided predictions :type predictions: ndarray :param predictions: Predictions of all models as a numpy array. :rtype: ndarray :return: Predicted classes.

ml_intuition.models.**get_model**(*model_key*, ***kwargs*)
Get a given instance of model specified by model_key.

Parameters

- **model_key** (str) – Specifies which model to use.

- **kwargs** – Any keyword arguments that the model accepts.

`ml_intuition.models.`**`model_2d`**(*kernel_size*, *n_kernels*, *n_layers*, *input_size*, *n_classes*, ***kwargs*)

2D model which consists of 2D convolutional blocks.

Parameters

- **kernel_size** (`int`) – Size of the convolutional kernel.
- **n_kernels** (`int`) – Number of kernels, i.e., the activation maps in each layer.
- **n_layers** (`int`) – Number of layers in the network.
- **input_size** (`int`) – Number of input channels, i.e., the number of spectral bands.
- **n_classes** (`int`) – Number of classes.
- **kwargs** – Additional arguments.

Return type `Sequential`

Returns Compiled model ready for training

`ml_intuition.models.`**`model_3d_deep`**(*n_classes*, *input_size*, ***kwargs*)

Deep 3D convolutional model from: https://arxiv.org/pdf/1907.11935.pdf

Parameters

- **n_classes** (`int`) – Number of classes in the dataset
- **input_size** (`int`) – Size of the input sample
- **kwargs** – Additional arguments.

Return type `Sequential`

Returns Compiled model ready for training

`ml_intuition.models.`**`model_3d_mfl`**(*kernel_size*, *n_kernels*, *n_classes*, *input_size*, ***kwargs*)

Multiple Features Learning model from: https://ieeexplore.ieee.org/document/6882821/figures#figures

Parameters

- **kernel_size** (`int`) – Size of the convolutional kernel
- **n_kernels** (`int`) – Number of kernels in each convolutional layer
- **n_classes** (`int`) – Number of classes in the dataset
- **input_size** (`int`) – Size of the input sample
- **kwargs** – Additional arguments.

Return type `Sequential`

Returns Compiled model ready for training

`ml_intuition.models.`**`pool_model_2d`**(*kernel_size*, *n_kernels*, *n_layers*, *input_size*, *n_classes*, ***kwargs*)

2D model which consists of 2D convolutional layers and 2D pooling layers.

Parameters

- **kernel_size** (`int`) – Size of the convolutional kernel.
- **n_kernels** (`int`) – Number of kernels, i.e., the activation maps in each layer.
- **n_layers** (`int`) – Number of layers in the network.
- **input_size** (`int`) – Number of input channels, i.e., the number of spectral bands.

- **n_classes** (`int`) – Number of classes.
- **kwargs** – Additional arguments.

Return type `Sequential`

Returns Compiled model ready for training

`ml_intuition.models.`**`unmixing_cube_based_cnn`**(*n_classes*, *input_size*, ***kwargs*)

Model for cube-based supervised hyperspectral unmixing proposed in the following publication (Chicago style citation):

Zhang, Xiangrong, Yujia Sun, Jingyan Zhang, Peng Wu, and Licheng Jiao. "Hyperspectral unmixing via deep convolutional neural networks." IEEE Geoscience and Remote Sensing Letters 15, no. 11 (2018): 1755-1759.

Parameters

- **n_classes** (`int`) – Number of classes.
- **input_size** (`int`) – Number of input spectral bands.
- **kwargs** – Additional arguments.

Return type `Sequential`

Returns Model proposed in the publication listed above.

`ml_intuition.models.`**`unmixing_cube_based_dcae`**(*n_classes*, *input_size*, ***kwargs*)

Model for cube-based unsupervised hyperspectral unmixing proposed in the following publication (Chicago style citation):

Khajehrayeni, Farshid, and Hassan Ghassemian. "Hyperspectral unmixing using deep convolutional autoencoders in a supervised scenario." IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 13 (2020): 567-576.

Parameters

- **n_classes** (`int`) – Number of classes.
- **input_size** (`int`) – Number of input spectral bands.
- **kwargs** – Additional arguments.

Return type `Sequential`

Returns Model proposed in the publication listed above.

`ml_intuition.models.`**`unmixing_pixel_based_cnn`**(*n_classes*, *input_size*, ***kwargs*)

Model for pixel-based supervised hyperspectral unmixing proposed in the following publication (Chicago style citation):

Zhang, Xiangrong, Yujia Sun, Jingyan Zhang, Peng Wu, and Licheng Jiao. "Hyperspectral unmixing via deep convolutional neural networks." IEEE Geoscience and Remote Sensing Letters 15, no. 11 (2018): 1755-1759.

Parameters

- **n_classes** (`int`) – Number of classes.
- **input_size** (`int`) – Number of input spectral bands.
- **kwargs** – Additional arguments.

Return type `Sequential`

Returns Model proposed in the publication listed above.

`ml_intuition.models.`**`unmixing_pixel_based_dcae`**(*n_classes*, *input_size*, ***kwargs*)

Model for pixel-based unsupervised hyperspectral unmixing proposed in the following publication (Chicago style citation):

Khajehrayeni, Farshid, and Hassan Ghassemian. "Hyperspectral unmixing using deep convolutional autoencoders in a supervised scenario." IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 13 (2020): 567-576.

Parameters

- **`n_classes`** (`int`) – Number of classes.
- **`input_size`** (`int`) – Number of input spectral bands.
- **`kwargs`** – Additional arguments.

Return type `Sequential`

Returns Model proposed in the publication listed above.

`ml_intuition.models.`**`unmixing_rnn_supervised`**(*n_classes*, ***kwargs*)

Model for the unmixing which utilizes a recurrent neural network (RNN) for extracting valuable information from the spectral domain in an supervised manner.

Parameters

- **`n_classes`** (`int`) – Number of classes.
- **`kwargs`** – Additional arguments.

Return type `Sequential`

Returns RNN model instance.

2.2.4 Module contents

2.3 scripts package

2.3.1 Subpackages

scripts.ensemble package

Submodules

scripts.ensemble.ensemble_runner module

Run ensemble using N number of different models. Models might use datasets preprocessed in various ways.

`scripts.ensemble.ensemble_runner.`**`run_experiments`**`(`*, *data_file_paths*, *train_size*, *val_size=0.1*, *stratified=True*, *background_label=0*, *channels_idx=0*, *neighborhood_sizes*, *save_data=False*, *n_runs*, *dest_path*, *model_paths*, *model_experiment_names*, *n_classes*, *voting='hard'*, *voting_model=None*, *voting_model_params=None*, *batch_size=1024*, *post_noise_sets*, *post_noise*, *noise_params=None*, *use_mlflow=False*, *experiment_name=None*, *run_name=None*, *use_unmixing=False*, *gt_file_paths*, *sub_test_size=None*`)`

Function for running experiments given a set of hyper parameters.

Parameters

- **`data_file_paths`** (`list[str]`) – Path to the data file. Supported types are: .npy
- **`train_size`** (`Union[int, float]`) – If float, should be between 0.0 and 1.0. If stratified = True, it represents percentage of each class to be extracted, If float and stratified = False, it represents percentage of the whole dataset to be extracted with samples drawn randomly, regardless of their class. If int and stratified = True, it represents number of samples to be drawn from each class. If int and stratified = False, it represents overall number of samples to be drawn regardless of their class, randomly. Defaults to 0.8
- **`val_size`** (`float`) – Should be between 0.0 and 1.0. Represents the percentage of each class from the training set to be extracted as a validation set, defaults to 0.1
- **`stratified`** (`bool`) – Indicated whether the extracted training set should be stratified, defaults to True
- **`background_label`** (`int`) – Label indicating the background in GT file
- **`channels_idx`** (`int`) – Index specifying the channels position in the provided data.
- **`neighborhood_sizes`** (`list[str]`) – List of sizes of neighborhoods of provided models.
- **`save_data`** (`bool`) – Whether to save the prepared dataset
- **`n_runs`** (`int`) – Number of total experiment runs.
- **`dest_path`** (`str`) – Path to where all experiment runs will be saved as subfolders in this directory.
- **`model_paths`** (`list[str]`) – Paths to all models to be used in ensemble
- **`model_experiment_names`** (`list[str]`) – Names of MLFlow experiments
- **`n_classes`** (`int`) – Number of classes.
- **`voting`** (`str`) – Method of ensemble voting. If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities. Else if 'booster', employs a new model, which is trained on the ensemble predictions on the training set.
- **`voting_model`** (`Optional[str]`) – Type of model to use when the voting argument is set to "booster". Supported models are: SVR, SVC, RFR, RFC, DTR, DTC.

- **voting_model_params** (Optional[str]) – Parameters of the voting model, same as the parameters for the noise injection.
- **batch_size** (int) – Size of the batch for the inference
- **post_noise_sets** (*list[str]*) – The list of sets to which the noise will be injected. One element can either be "train", "val" or "test".
- **post_noise** (*list[str]*) – The list of names of noise injection methods after the normalization transformations.
- **noise_params** (Optional[str]) – JSON containing the parameter setting of injection methods. Exemplary value for this parameter: "{"mean": 0, "std": 1, "pa": 0.1}". This JSON should include all parameters for noise injection functions that are specified in pre_noise and post_noise arguments. For the accurate description of each parameter, please refer to the ml_intuition/data/noise.py module.
- **use_mlflow** (bool) – Whether to log metrics and artifacts to mlflow.
- **experiment_name** (Optional[str]) – Name of the experiment. Used only if use_mlflow = True
- **run_name** (Optional[str]) – Name of the run. Used only if use_mlflow = True.
- **use_unmixing** (bool) – Boolean indicating whether to utilize the unmixing functionality.
- **gt_file_paths** (*list[str]*) – Path to the ground-truth data files. Supported types are: .npy
- **sub_test_size** (Optional[int]) – Number of pixels to subsample from the test set instead of performing the inference on all untrained samples.

scripts.ensemble.evaluate_with_ensemble module

Evaluate the dataset using an ensemble of models.

`scripts.ensemble.evaluate_with_ensemble.`**`evaluate`**`(`*, *y_pred*, *data*, *dest_path*, *n_classes*, *model_path*, *voting='hard'*, *train_set_predictions=None*, *voting_model=None*, *voting_model_params=None*`)`

Function for evaluating the trained model.

Parameters

- **y_pred** (ndarray) – Predictions of test set.
- **model_path** (str) – Path to the model.
- **data** – Either path to the input data or the data dict.
- **dest_path** (str) – Directory in which to store the calculated metrics
- **n_classes** (int) – Number of classes.
- **voting** (str) – Method of ensemble voting. If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities. Else if 'booster', employs a new model, which is trained on the ensemble predictions on the training set.
- **train_set_predictions** (Optional[ndarray]) – Predictions of the train set. Only used if 'voting' = 'classifier'.

- **voting_model** (Optional[str]) – Type of model to use when the voting argument is set to "booster". This indicates, that a new model is trained on the ensemble predictions on the learning set, to leverage the quality of the classification or regression. Supported models are: SVR, SVC (support vector machine for regression and classification), RFR, RFC (random forest for regression and classification), DTR, DTC (decision tree for regression and classification).
- **voting_model_params** (Optional[str]) – Parameters of the voting model, should be specified in the same manner as the parameters for the noise injection.

Module contents

scripts.unmixing package

Submodules

scripts.unmixing.evaluate_unmixing module

Perform the inference of the unmixing model on the test dataset.

`scripts.unmixing.evaluate_unmixing.`**`evaluate`**(*data*, *model_path*, *dest_path*, *neighborhood_size*, *batch_size*, *endmembers_path*, *use_ensemble=False*, *ensemble_copies=1*, *noise_params=None*, *voting='mean'*, *voting_model=None*, *voting_model_params=None*, *seed=0*)

Function for evaluating the trained model for the unmixing problem.

Parameters

- **model_path** (str) – Path to the model.
- **data** – Either path to the input data or the data dict.
- **dest_path** (str) – Path to the directory to store the calculated metrics.
- **neighborhood_size** (int) – Size of the spatial patch.
- **batch_size** (int) – Size of the batch for inference.
- **endmembers_path** (str) – Path to the endmembers file containing average reflectances for each class. Used only when use_unmixing is set to True.
- **use_ensemble** (bool) – Boolean indicating whether to use ensembles functionality.
- **ensemble_copies** (int) – Number of copies of the original model to create.
- **noise_params** (Optional[str]) – Parameters for the noise when creating copies of the base model. Those can be for instance the mean, or standard deviation of the noise.
- **voting** (str) – Method of ensemble voting. If 'booster', employs a new model, which is trained on the ensemble predictions on the training set. Else if 'mean', averages the predictions of all models, without any weights.
- **voting_model** (Optional[str]) – Type of the model to use when the voting argument is set to 'booster'. This indicates, that a new model is trained on the ensemble's predictions on the learning set, to leverage the quality of the regression. Supported models are: SVR (support vector machine for regression), RFR (random forest for regression) and DTR (decision tree for regression).

- **voting_model_params** (`Optional[str]`) – Parameters of the voting model. Used only when the type of voting is set to 'booster'. Should be specified analogously to the noise injection parameters in the 'noise' module.
- **seed** (`int`) – Parameter used for the experiments reproduction.

scripts.unmixing.train_unmixing module

Perform the training of the models for the unmixing problem.

`scripts.unmixing.train_unmixing.`**`train`**(*data*, *model_name*, *dest_path*, *sample_size*, *n_classes*, *neighborhood_size*, *lr*, *batch_size*, *epochs*, *verbose*, *shuffle*, *patience*, *endmembers_path*, *seed*)

Function for running experiments on various unmixing models, given a set of hyper parameters.

Parameters

- **data** (`Dict[str, ndarray]`) – The data dictionary containing the subsets for training and validation. First dimension of the dataset should be the number of samples.
- **model_name** (`str`) – Name of the model, it serves as a key in the dictionary holding all functions returning models.
- **dest_path** (`str`) – Path to where all experiment runs will be saved as subdirectories in this directory.
- **sample_size** (`int`) – Spectral size of the input sample.
- **n_classes** (`int`) – Number of classes.
- **neighborhood_size** (`int`) – Size of the spatial patch.
- **lr** (`float`) – Learning rate for the model i.e., it regulates the size of the step in the gradient descent process.
- **batch_size** (`int`) – Size of the batch used in training phase, it is the number of samples per gradient step.
- **epochs** (`int`) – Number of epochs for the model to train.
- **verbose** (`int`) – Verbosity mode used in training, (0, 1 or 2).
- **shuffle** (`bool`) – Boolean indicating whether to shuffle the dataset.
- **patience** (`int`) – Number of epochs without improvement in order to stop the training phase.
- **endmembers_path** (`str`) – Path to the endmembers file containing the average reflectances for each class i.e., the pure spectra. Used only when use_unmixing is set to True.
- **seed** (`int`) – Seed for experiment reproducibility.

scripts.unmixing.unmixing_experiments_runner module

Run experiments given set of hyperparameters for the unmixing problem.

`scripts.unmixing.unmixing_experiments_runner.`**`run_experiments`**`(`*, *data_file_path*, *ground_truth_path=None*, *train_size*, *val_size=0.1*, *sub_test_size=None*, *channels_idx=-1*, *neighborhood_size=None*, *n_runs=1*, *model_name*, *save_data=0*, *dest_path=None*, *sample_size*, *n_classes*, *lr=None*, *batch_size=256*, *epochs=100*, *verbose=2*, *shuffle=True*, *patience=15*, *use_mlflow=False*, *endmembers_path=None*, *experiment_name=None*, *run_name=None*`)`

Function for running experiments on unmixing given a set of hyperparameters.

Parameters

- **`data_file_path`** (`str`) – Path to the data file. Supported types are: .npy.
- **`ground_truth_path`** (`Optional[str]`) – Path to the ground-truth data file.
- **`train_size`** (*`Union[int, float]`*) – If float, should be between 0.0 and 1.0, if int, it represents number of samples to draw from data.
- **`val_size`** (`float`) – Should be between 0.0 and 1.0. Represents the percentage of samples to extract from the training set.
- **`sub_test_size`** (`Optional[int]`) – Number of pixels to subsample the test set instead of performing the inference on the entire subset.
- **`channels_idx`** (`int`) – Index specifying the channels position in the provided data.
- **`neighborhood_size`** (`Optional[int]`) – Size of the spatial patch.
- **`save_data`** (`bool`) – Boolean indicating whether to save the prepared dataset.
- **`n_runs`** (`int`) – Number of total experiment runs.
- **`model_name`** (`str`) – Name of the model, it serves as a key in the dictionary holding all functions returning models.
- **`dest_path`** (`Optional[str]`) – Path to the directory where all experiment runs will be saved as subdirectories.

- **sample_size** (int) – Spectral size of the input sample.
- **n_classes** (int) – Number of classes.
- **lr** (Optional[float]) – Learning rate for the model i.e., it regulates the size of the step in the gradient descent process.
- **batch_size** (int) – Size of the batch used in training phase, it is the number of samples to utilize per single gradient step.
- **epochs** (int) – Total number of epochs for model to train.
- **verbose** (int) – Verbosity mode used in training, (0, 1 or 2).
- **shuffle** (bool) – Boolean indicating whether to shuffle dataset.
- **patience** (int) – Number of epochs without improvement in order to stop the training phase.
- **use_mlflow** (bool) – Boolean indicating whether to log metrics and artifacts to mlflow.
- **endmembers_path** (Optional[str]) – Path to the endmembers file containing the average reflectances for each class. Used only when 'use_unmixing' is set to True.
- **experiment_name** (Optional[str]) – Name of the experiment. Used only if 'use_mlflow' is set to True.
- **run_name** (Optional[str]) – Name of the run. Used only if 'use_mlflow' is set to True.

Module contents

scripts.quantization package

Submodules

scripts.quantization.freeze_model module

Freeze the graph and store it in .pb (Protocol Buffers) format

scripts.quantization.freeze_model.**main**(*, *model_path*, *output_dir*)

Freeze the graph and store it in .pb (Protocol Buffers) format

Parameters

- **model_path** (str) – Path to the model to be saved
- **output_dir** (str) – Directory in which the .pb graph and nodes json will be stored

scripts.quantization.quantize_runner module

Freeze model, quantize it and evaluate N times.

`scripts.quantization.quantize_runner.`**`run_experiments`**`(`*, *input_dir*, *n_runs*, *dest_path*, *data_file_path=None*, *ground_truth_path=None*, *dataset_path=None*, *background_label=0*, *channels_idx=2*, *channels_count=103*, *train_size*, *batch_size=64*, *stratified=True*, *gpu=0*`)`

Freeze model, quantize it and evaluate N times.

Parameters

- **`input_dir`** (`str`) – Directory with saved data and models, each in separate *experiment_n* folder.
- **`n_runs`** (`int`) – Number of total experiment runs.
- **`dest_path`** (`str`) – Path to where all experiment runs will be saved as sub folders in this directory.
- **`data_file_path`** (`Optional[str]`) – Path to the data file. Supported types are: .npy and .md5. This is optional, if the data is not already saved in the input_dir.
- **`ground_truth_path`** (`Optional[str]`) – Path to the data file.
- **`dataset_path`** (`Optional[str]`) – Path to the already extracted .h5 dataset
- **`background_label`** (`int`) – Label indicating the background in GT file
- **`channels_idx`** (`int`) – Index specifying the channels position in the provided data
- **`channels_count`** (`int`) – Number of channels (bands) in the image.
- **`train_size`** (*`Union[int, float]`*) – If float, should be between 0.0 and 1.0. If stratified = True, it represents percentage of each class to be extracted, If float and stratified = False, it represents percentage of the whole dataset to be extracted with samples drawn randomly, regardless of their class. If int and stratified = True, it represents number of samples to be drawn from each class. If int and stratified = False, it represents overall number of samples to be drawn regardless of their class, randomly. Defaults to 0.8
- **`stratified`** (`bool`) – Indicated whether the extracted training set should be stratified, defaults to True
- **`batch_size`** (`int`) – Batch size
- **`gpu`** (`bool`) – Whether to run quantization on gpu.

Module contents

2.3.2 Submodules

2.3.3 scripts.artifacts_reporter module

Collect the artifacts report based on the experiment runs placed in the “experiments_path” directory.

`scripts.artifacts_reporter.`**`collect_artifacts_report`**`(`*, *experiments_path*, *dest_path*, *filename=None*, *use_mlflow=False*`)`

Collect the artifacts report based on the experiment runs placed in the “experiments_path” directory.

Parameters

- **experiments_path** (str) – Path to the directory containing the experiment subdirectories.
- **dest_path** (str) – Path to the destination directory or full path to the report .csv file.
- **filename** (Optional[str]) – Name of the file holding metrics. Defaults to 'inference_metrics.csv'.
- **use_mlflow** (bool) – Whether to log metrics and artifacts to mlflow.

2.3.4 scripts.evaluate_model module

Perform the inference of the model on the provided dataset.

scripts.evaluate_model.**evaluate**(*, *data*, *model_path*, *dest_path*, *n_classes*, *batch_size=1024*, *use_ensemble=False*, *ensemble_copies=1*, *voting='hard'*, *noise*, *noise_sets*, *noise_params=None*, *seed=0*)

Function for evaluating the trained model.

Parameters

- **model_path** (str) – Path to the model.
- **data** – Either path to the input data or the data dict.
- **dest_path** (str) – Directory in which to store the calculated metrics.
- **n_classes** (int) – Number of classes.
- **batch_size** (int) – Size of the batch for inference.
- **use_ensemble** (bool) – Use ensemble for prediction.
- **ensemble_copies** (int) – Number of model copies for the ensemble.
- **voting** (str) – Method of ensemble voting. If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities.
- **noise** (*list[str]*) – List containing names of used noise injection methods that are performed after the normalization transformations.
- **noise_sets** (*list[str]*) – List of sets that are affected by the noise injection. For this module single element can be "test".
- **noise_params** (Optional[str]) – JSON containing the parameters setting of noise injection methods. Exemplary value for this parameter: "{"mean": 0, "std": 1, "pa": 0.1}". This JSON should include all parameters for noise injection functions that are specified in the noise argument. For the accurate description of each parameter, please refer to the ml_intuition/data/noise.py module.
- **seed** (int) – Seed for RNG.

2.3.5 scripts.experiments_runner module

Run training and evaluation of a model N times, given set of hyperparameters. Has the option to inject noise into train, val and test set.

`scripts.experiments_runner.`**`run_experiments`**`(`*, *data_file_path*, *ground_truth_path=None*, *train_size*, *val_size=0.1*, *stratified=True*, *background_label=0*, *channels_idx=0*, *neighborhood_size=None*, *n_runs*, *model_name*, *kernel_size=5*, *n_kernels=200*, *save_data=0*, *n_layers=1*, *dest_path=None*, *sample_size*, *n_classes*, *lr=0.001*, *batch_size=128*, *epochs=200*, *verbose=2*, *shuffle=True*, *patience=15*, *pre_noise*, *pre_noise_sets*, *post_noise*, *post_noise_sets*, *noise_params=None*, *use_mlflow=False*, *experiment_name=None*, *run_name=None*`)`

Function for running experiments given a set of hyper parameters.

Parameters

- **`data_file_path`** (`str`) – Path to the data file. Supported types are: .npy.
- **`ground_truth_path`** (`Optional[str]`) – Path to the ground-truth data file.
- **`train_size`** (*`Union[int, float]`*) – If float, should be between 0.0 and 1.0. If stratified = True, it represents percentage of each class to be extracted. If float and stratified = False, it represents percentage of the whole dataset to be extracted with samples drawn randomly, regardless of their class. If int and stratified = True, it represents number of samples to be drawn from each class. If int and stratified = False, it represents overall number of samples to be drawn regardless of their class, randomly. Defaults to 0.8.
- **`val_size`** (`float`) – Should be between 0.0 and 1.0. Represents the percentage of each class from the training set to be extracted as a validation set. Defaults to 0.1.
- **`stratified`** (`bool`) – Indicated whether the extracted training set should be stratified. Defaults to True.
- **`background_label`** (`int`) – Label indicating the background in GT file.
- **`channels_idx`** (`int`) – Index specifying the channels position in the provided data.
- **`neighborhood_size`** (`Optional[int]`) – Size of the spatial patch.
- **`save_data`** (`bool`) – Whether to save the prepared dataset.
- **`n_runs`** (`int`) – Number of total experiment runs.
- **`model_name`** (`str`) – Name of the model, it serves as a key in the dictionary holding all functions returning models.
- **`kernel_size`** (`int`) – Size of ech kernel in each layer.
- **`n_kernels`** (`int`) – Number of kernels in each layer.
- **`n_layers`** (`int`) – Number of layers in the model.
- **`dest_path`** (`Optional[str]`) – Path to where all experiment runs will be saved as subfolders in this directory.
- **`sample_size`** (`int`) – Size of the input sample.
- **`n_classes`** (`int`) – Number of classes.

- **lr** (float) – Learning rate for the model, i.e., regulates the size of the step in the gradient descent process.
- **batch_size** (int) – Size of the batch used in training phase, it is the size of samples per gradient step.
- **epochs** (int) – Number of epochs for model to train.
- **verbose** (int) – Verbosity mode used in training, (0, 1 or 2).
- **shuffle** (bool) – Boolean indicating whether to shuffle dataset dataset_key each epoch.
- **patience** (int) – Number of epochs without improvement in order to stop the training phase.
- **pre_noise** (*list[str]*) – The list of names of noise injection methods before the normalization transformations. Exemplary names are "gaussian" or "impulsive".
- **pre_noise_sets** (*list[str]*) – The list of sets to which the noise will be injected. One element can either be "train", "val" or "test".
- **post_noise** (*list[str]*) – The list of names of noise injection methods after the normalization transformations.
- **post_noise_sets** (*list[str]*) – The list of sets to which the noise will be injected.
- **noise_params** (Optional[str]) – JSON containing the parameter setting of injection methods. Exemplary value for this parameter: "{"mean": 0, "std": 1, "pa": 0.1}". This JSON should include all parameters for noise injection functions that are specified in pre_noise and post_noise arguments. For the accurate description of each parameter, please refer to the ml_intuition/data/noise.py module.
- **use_mlflow** (bool) – Whether to log metrics and artifacts to mlflow.
- **experiment_name** (Optional[str]) – Name of the experiment. Used only if use_mlflow = True
- **run_name** (Optional[str]) – Name of the run. Used only if use_mlflow = True.

2.3.6 scripts.inference_runner module

Run inference N times on the provided model given set of hyperparameters. Has the option to inject noise into the test set.

scripts.inference_runner.**run_experiments**(*, *data_file_path=None*, *ground_truth_path=None*, *dataset_path=None*, *train_size*, *val_size=0.1*, *stratified=True*, *background_label=0*, *channels_idx=0*, *neighborhood_size=None*, *save_data=False*, *n_runs*, *dest_path*, *models_path*, *model_name='model_2d'*, *n_classes*, *use_ensemble=False*, *ensemble_copies=None*, *voting='hard'*, *batch_size=1024*, *post_noise_sets*, *post_noise*, *noise_params=None*, *use_mlflow=False*, *experiment_name=None*, *model_exp_name=None*, *run_name=None*)

Run inference on the provided model given set of hyperparameters.

Parameters

- **data_file_path** (Optional[str]) – Path to the data file. Supported types are: .npy

- **ground_truth_path** (Optional[str]) – Path to the ground-truth data file.
- **dataset_path** (Optional[str]) – Path to the already extracted .h5 dataset
- **train_size** (*Union[int, float]*) – If float, should be between 0.0 and 1.0. If stratified = True, it represents percentage of each class to be extracted, If float and stratified = False, it represents percentage of the whole dataset to be extracted with samples drawn randomly, regardless of their class. If int and stratified = True, it represents number of samples to be drawn from each class. If int and stratified = False, it represents overall number of samples to be drawn regardless of their class, randomly. Defaults to 0.8
- **val_size** (float) – Should be between 0.0 and 1.0. Represents the percentage of each class from the training set to be extracted as a validation set. Defaults to 0.1.
- **stratified** (bool) – Indicated whether the extracted training set should be stratified. Defaults to True.
- **background_label** (int) – Label indicating the background in GT file.
- **channels_idx** (int) – Index specifying the channels position in the provided data.
- **neighborhood_size** (Optional[int]) – Size of the neighbourhood for the model.
- **save_data** (bool) – Whether to save the prepared dataset
- **n_runs** (int) – Number of total experiment runs.
- **dest_path** (str) – Path to where all experiment runs will be saved as subfolders in this directory.
- **models_path** (str) – Name of the model, it serves as a key in the dictionary holding all functions returning models.
- **model_name** (str) – The name of model for the inference.
- **n_classes** (int) – Number of classes.
- **use_ensemble** (bool) – Use ensemble for prediction.
- **ensemble_copies** (Optional[int]) – Number of model copies for the ensemble.
- **voting** (str) – Method of ensemble voting. If 'hard', uses predicted class labels for majority rule voting. Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities.
- **batch_size** (int) – Size of the batch for the inference
- **post_noise_sets** (*list[str]*) – The list of sets to which the noise will be injected. One element can either be "train", "val" or "test".
- **post_noise** (*list[str]*) – The list of names of noise injection methods after the normalization transformations.
- **noise_params** (Optional[str]) – JSON containing the parameter setting of injection methods. Exemplary value for this parameter: "{"mean": 0, "std": 1, "pa": 0.1}". This JSON should include all parameters for noise injection functions that are specified in pre_noise and post_noise arguments. For the accurate description of each parameter, please refer to the ml_intuition/data/noise.py module.
- **use_mlflow** (bool) – Whether to log metrics and artifacts to mlflow.
- **experiment_name** (Optional[str]) – Name of the experiment. Used only if use_mlflow = True.
- **run_name** (Optional[str]) – Name of the run. Used only if use_mlflow = True.

2.3.7 scripts.predict_with_model module

Perform the inference of the model on the dataset and return predictions.

`scripts.predict_with_model.`**`predict`**`(`*, *data*, *model_path*, *batch_size=1024*, *dataset_to_predict='test'*, *use_unmixing=False*, *neighborhood_size=None*`)`

Function for evaluating the trained model.

Parameters

- **`data`** – Either path to the input data or the data dict.
- **`model_path`** (`str`) – Path to the model.
- **`batch_size`** (`int`) – Size of the batch for inference
- **`dataset_to_predict`** (`str`) – Name of the dataset to predict, ‘train’ or ‘test’.
- **`use_unmixing`** (`bool`) – Boolean indicating whether to use unmixing functionality.
- **`neighborhood_size`** (`Optional[int]`) – Size of the local spatial extend of each sample.

2.3.8 scripts.prepare_data module

Load the data, reformat it to have [SAMPLES, ...] dimensions, split it into train, test and val sets and save them in .h5 file with ‘train’, ‘val’ and ‘test’ groups, each having ‘data’ and ‘labels’ keys.

`scripts.prepare_data.`**`main`**`(`*, *data_file_path*, *ground_truth_path*, *output_path=None*, *train_size*, *val_size=0.1*, *stratified=True*, *background_label=0*, *neighborhood_size=None*, *channels_idx=0*, *save_data=False*, *seed=0*, *use_unmixing=False*`)`

Parameters

- **`data_file_path`** (`str`) – Path to the data file. Supported types are: .npy.
- **`ground_truth_path`** (`str`) – Path to the data file.
- **`output_path`** (`Optional[str]`) – Path under in which the output .h5 file will be stored. Used only if the parameter save_data is set to True.
- **`train_size`** (*`float or int`*) – If float, should be between 0.0 and 1.0. If stratified = True, it represents percentage of each class to be extracted, If float and stratified = False, it represents percentage of the whole dataset to be extracted with samples drawn randomly, regardless of their class. If int and stratified = True, it represents number of samples to be drawn from each class. If int and stratified = False, it represents overall number of samples to be drawn regardless of their class, randomly. Defaults to 0.8
- **`val_size`** (`float`) – Should be between 0.0 and 1.0. Represents the percentage of each class from the training set to be extracted as a validation set. Defaults to 0.1.
- **`stratified`** (`bool`) – Indicated whether the extracted training set should be stratified. Defaults to True.
- **`background_label`** (`int`) – Label indicating the background in GT file.
- **`neighborhood_size`** (`Optional[int]`) – Neighborhood size of the pixel to extract along with its spectral bands. Use only if you are training 2D or 3D convolutional model.
- **`channels_idx`** (`int`) – Index specifying the channels position in the provided data.

- **save_data** (bool) – Whether to save data as .md5 or to return it as a dict.
- **seed** (int) – Seed used for data shuffling.
- **use_unmixing** (bool) – Boolean indicating whether to perform experiments on the unmixing datasets, where classes in each pixel are present as fractions.

Raises **TypeError** – When provided data or labels file is not supported.

2.3.9 scripts.train_model module

Perform the training of the model. Has the option to inject noise into train and val set.

scripts.train_model.**train**(**, data, model_name, dest_path, sample_size, n_classes, kernel_size=3, n_kernels=16, n_layers=1, lr=0.005, batch_size=150, epochs=10, verbose=2, shuffle=True, patience=3, seed=0, noise, noise_sets, noise_params=None*)

Function for training tensorflow models given a dataset.

Parameters

- **model_name** (str) – Name of the model, it serves as a key in the dictionary holding all functions returning models.
- **kernel_size** (int) – Size of ech kernel in each layer.
- **n_kernels** (int) – Number of kernels in each layer.
- **n_layers** (int) – Number of layers in the model.
- **dest_path** (str) – Path to where to save the model under the name "model_name".
- **sample_size** (int) – Size of the input sample.
- **n_classes** (int) – Number of classes.
- **lr** (float) – Learning rate for the model, i.e., regulates the size of the step in the gradient descent process.
- **data** – Either path to the input data or the data dict itself. First dimension of the dataset should be the number of samples.
- **batch_size** (int) – Size of the batch used in training phase, it is the size of samples per gradient step.
- **epochs** (int) – Number of epochs for model to train.
- **verbose** (int) – Verbosity mode used in training, (0, 1 or 2).
- **shuffle** (bool) – Boolean indicating whether to shuffle dataset dataset_key each epoch.
- **patience** (int) – Number of epochs without improvement in order to stop the training phase.
- **seed** (int) – Seed for training reproducibility.
- **noise** (*list[str]*) – List containing names of used noise injection methods that are performed after the normalization transformations.
- **noise_sets** (*list[str]*) – List of sets that are affected by the noise injection methods. For this module single element can be either "train" or "val".
- **noise_params** (Optional[str]) – JSON containing the parameters setting of injection methods. Exemplary value for this parameter: "{"mean": 0, "std": 1, "pa": 0.1}". This JSON should include all parameters for noise injection functions that are specified

in the noise argument. For the accurate description of each parameter, please refer to the ml_intuition/data/noise.py module.

2.3.10 Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX