

---

# **Beetles**

***Release 1.5***

**KP Labs**

**Jun 17, 2021**



**CONTENTS:**

<b>1</b>	<b>Installation</b>	<b>1</b>
1.1	Core install . . . . .	1
1.2	Standard Tensorflow install . . . . .	1
1.3	Xilinx toolbox install . . . . .	2
<b>2</b>	<b>machine-learning</b>	<b>3</b>
2.1	cloud_detection package . . . . .	3
2.2	ml_intuition package . . . . .	23
2.3	scripts package . . . . .	37
<b>3</b>	<b>Indices and tables</b>	<b>53</b>
	<b>Python Module Index</b>	<b>55</b>
	<b>Index</b>	<b>57</b>



## INSTALLATION

### 1.1 Core install

- 1) Make sure you have Python 3.6 installed. It can be downloaded and installed from <https://www.python.org/downloads/>
- 2) Make sure you have conda installed. Full guide for your operating system is available here: <https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html>
- 3) Copy the repo <https://github.com/ESA-PhiLab/hypernet> onto your computer and enter the *hypernet/beetles* directory.
- 4) To create the environment with all the required dependencies, execute the following line:

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

- 5) Activate the environment by running:

```
conda activate decent
```

### 1.2 Standard Tensorflow install

If you are not interested in the model quantization and compilation, you can proceed with the following steps. However, if you are, please move to the next section.

If you are going to run your code on CPU, execute:

```
pip install tensorflow==1.12
```

or for the GPU:

```
pip install tensorflow-gpu==1.12
```

## 1.3 Xilinx toolbox install

The Xilinx toolbox is required to run the model quantization and compilation. As of now, it is only required for the *model\_quantization\_xilinx* example. So if you are not interested in that, you can skip this part.

- 1) For this part, you need **Ubuntu** 14.04, 16.04 or 18.04.
- 2) Download the *xilinx\_dnndk\_v3.1\_190809.tar.gz* from <https://www.xilinx.com/products/design-tools/ai-inference/ai-developer-hub.html#edge>
- 3) Execute

```
pip install ${TF_DNNDK_WHEEL_PATH}
```

Where *TF\_DNNDK\_WHEEL\_PATH* is a path to the tensorflow *.whl* package located in the directory you have just downloaded. Select the one that suits you, based on the version of the operating system, python version and whether you would like to run on CPU or GPU. Keep in mind that you will need CUDA and cuDNN libraries for the GPU. If you run into any problems, the more detailed installation process could be found here: [https://www.xilinx.com/support/documentation/sw\\_manuals/ai\\_inference/v1\\_6/ug1327-dnndk-user-guide.pdf](https://www.xilinx.com/support/documentation/sw_manuals/ai_inference/v1_6/ug1327-dnndk-user-guide.pdf)

- 4) To validate the installation, run:

```
decent_q --help
```

If you have completed the setup correctly, you are now able to run the code and the examples. The process of installing and running the jupyter notebooks is described in the official documentation: <https://jupyter.readthedocs.io/en/latest/install.html>

## MACHINE-LEARNING

## 2.1 cloud\_detection package

### 2.1.1 Subpackages

**cloud\_detection.scripts package**

**Submodules**

**cloud\_detection.scripts.cluster module**

Run K-Means and Gaussian Mixture clustering for provided data and evaluate the performance through the use of unsupervised metrics.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
cloud_detection.scripts.cluster.run_clustering(*, config_path, alg, dest_path,
                                              n_clusters_max, use_mlflow, experi-
                                              ment_name=None, run_name=None)
```

Run clustering for data specified in the experiment configuration.

#### Parameters

- **config\_path** (str) – Path to the configuration experiment file. It should be located in “cfg” directory and contain necessary keys, which specify the target dataset names and base path.
- **alg** (str) – Type of algorithm to utilize for clustering. Possible options are: “km” for K-Means and “gm” for Gaussian Mixture Model respectively.
- **dest\_path** (str) – Destination path where all results will be stored.
- **n\_clusters\_max** (int) – Maximum number of clusters to run on all images. Each will be clustered with the number of centres in range from two to “n\_clusters\_max”.
- **use\_mlflow** (bool) – Boolean indicating whether to utilize 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.

### cloud\_detection.scripts.generate\_training\_patches\_paths module

Script to generate to file training patches paths for all 5 experiments of cloud detection using RGBNir data. The generated file can be used to recreate the different training datasets used for all 5 experiments by loading appropriate training patches.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
cloud_detection.scripts.generate_training_patches_paths.generate (train_path=PosixPath('datasets/cloud-Cloud/38-Cloud_training'),  
                                                                train_size=0.8)
```

Generate to file training patches paths for all 5 experiments of cloud detection using RGBNir data.

#### Parameters

- **train\_path** (Path) – Path to 38-Cloud training dataset.
- **train\_size** (float) – Size of the training set (the rest is used for validation).

### cloud\_detection.scripts.panchromatic\_thresholding module

Perform panchromatic thresholding.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
class cloud_detection.scripts.panchromatic_thresholding.ThresholdingClassifier (thr_prop=0.1)  
    Bases: object
```

**Parameters** **thr\_prop** (float) – threshold to classify pixels into clouds, e.g. for thr\_prop = 0.5, min pixel value = 1, max pixel value = 5, all pixels with value greater or equal to 3 will be classified as clouds.

**fit** (X)

Fit method.

**Parameters** **X** (ndarray) – image for fitting.

**Returns** self.

**predict** (X)

Predict method.

**Parameters** **X** (ndarray) – image for predicting.

**Return type** ndarray

**Returns** cloud mask of the same shape as X, where 0-not cloud & 1-cloud.



```
cloud_detection.scripts.panchromatic_thresholding.run(thresholds=array([0. , 0.05,
                                0.1 , 0.15, 0.2 , 0.25, 0.3
                                , 0.35, 0.4 , 0.45, 0.5 ,
                                0.55, 0.6 , 0.65, 0.7 , 0.75,
                                0.8 , 0.85, 0.9 , 0.95, 1. ]),
                                dpath=PosixPath('datasets/clouds/Landsat-
                                Cloud-Cover-Assessment-
                                Validation-Data-Partial'),
                                rpath=PosixPath('artifacts'),
                                cpath=PosixPath('cloud_detection/cfg/exp_panchroma
                                band_num=8,          met-
                                ric_fns=[<function normal-
                                ized_mutual_info_score>,
                                <function          ad-
                                justed_rand_score>],
                                tf_metric_fns=[<cloud_detection.losses.JaccardIndex
                                object>,
                                <cloud_detection.losses.JaccardIndexMetric
                                object>,
                                <cloud_detection.losses.DiceCoefMetric
                                object>],          mlflow=False,
                                run_name=None)
```

Perform panchromatic thresholding for given threshold values.

#### Parameters

- **thresholds** (ndarray) – threshold values to perform panchromatic thresholding.
- **dpath** (Path) – path to dataset.
- **rpath** (Path) – path to directory where results and artifacts should be logged (randomly named directory will be created to store the results).
- **cpath** (Path) – path to experiment config file.
- **band\_num** (int) – number of the panchromatic band.
- **metric\_fns** (List[Callable]) – non-Tensorflow metric functions to run evaluation of the thresholding. Must be of the form func(labels\_true, labels\_pred).
- **tf\_metric\_fns** (List[Callable]) – TensorFlow metric functions to run evaluation of the thresholding. Must be of the form func(labels\_true, labels\_pred).
- **mlflow** (bool) – whether to use mlflow.
- **run\_name** (Optional[str]) – name of the run.

## 2.1.2 Submodules

### 2.1.3 cloud\_detection.data\_gen module

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

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
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 patches dimensions.
- **shuffle** (bool) – if True shuffles dataset before training and 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_paths, batch_size, data_part=(0.0, 1.0),
                                       with_gt=False, patch_size=384, bands=(4, 3, 2, 5),
                                       bands_names=('red', 'green', 'blue', 'nir'),
                                       resize=False, normalize=True, standardize=False,
                                       shuffle=True)
```

Bases: tensorflow.python.keras.utils.data\_utils.Sequence

Data generator for L8CCA clouds segmentation dataset. Works with Keras generators.

Prepare generator and init paths to files containing image channels.

#### Parameters

- **img\_paths** (List[Path]) – paths to the dirs containing L8CCA images files.
- **batch\_size** (int) – size of generated batches, only one batch is loaded to memory at a time.
- **data\_part** (Tuple[float]) – part of data to include, e.g., (0., 0.2) generates dataloader with samples up to 20-th percentile without 20-th percentile, while (0.3, 0.8) generates dataloader with samples from 30-th percentile (including it) up to 80-th percentile (without it). (x, 1.) is an exception to the rule, generating dataloader with samples from the x-th percentile (including it) up to AND INCLUDING the last datapoint. To include all samples, use (0., 1.). If shuffle=True, partitions dataset based on shuffled data (with a set seed), else partitions unshuffled dataset.
- **with\_gt** (bool) – whether to include groundtruth.
- **patch\_size** (int) – size of the patches.
- **bands** (Tuple[int]) – band numbers to load
- **bands\_names** (Tuple[str]) – names of the bands to load. Should have the same number of elements as bands.

- **resize** (bool) – whether to resize img to gt. If True and with\_gt=False, will load GT to infer its shape and then delete it.
- **normalize** (bool) – whether to normalize the image.
- **standardize** (bool) – whether to standardize the image.
- **shuffle** (bool) – if True shuffles dataset before training and on each epoch end, else returns dataloader sorted according to img\_paths order.

**on\_epoch\_end()**

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

`cloud_detection.data_gen.main_38Cloud()`

Demo 38Cloud data loading.

`cloud_detection.data_gen.main_L8CCA()`

Demo L8CCA data loading.

## 2.1.4 cloud\_detection.evaluate\_38Cloud module

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

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
cloud_detection.evaluate_38Cloud.evaluate_model(model, thr, dpath, gtpath, vpath,
                                                rpath, vids, batch_size, img_ids=None,
                                                metric_fns=[<function      normal-
                                                ized_mutual_info_score>, <function
                                                adjusted_rand_score>], 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 directory 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.
- **metric\_fns** (List[Callable]) – non-Tensorflow metric functions to run evaluation of the model. Must be of the form func(labels\_true, labels\_pred).
- **mlflow** (bool) – whether to use MLFlow.
- **run\_name** (Optional[str]) – name of the run.

**Return type** Tuple[Dict, List]

**Returns** evaluation metrics and evaluation times for scenes.

`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.

```
cloud_detection.evaluate_38Cloud.run_evaluation(*, model_hash, thr=0.5,
                                                dpath='datasets/clouds/38-Cloud/38-Cloud_test',
                                                gtpath='datasets/clouds/38-Cloud/38-Cloud_test/Entire_scene_gts',
                                                vpath='datasets/clouds/38-Cloud/38-Cloud_test/Natural_False_Color',
                                                rpath='artifacts/', vids, batch_size=32,
                                                img_ids, mlflow=False,
                                                run_name=None)
```

Load model given model hash and get evaluation metrics on 38-Cloud testset.

#### Parameters

- **model\_hash** (str) – MLFlow hash of the model to load.
- **thr** (float) – threshold to be used during evaluation.
- **dpath** (str) – path to dataset.
- **gtpath** (str) – path to dataset ground truths.
- **vpath** (str) – path to dataset (false color) visualisation images.
- **rpath** (str) – path to directory 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** (list[int]) – if given, process only these images.
- **mlflow** (bool) – whether to use MLFlow.
- **run\_name** (Optional[str]) – name of the run.

## 2.1.5 cloud\_detection.evaluate\_L8CCA module

Get evaluation metrics for given model on L8CCA dataset.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
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,
                                              bands=(4, 3, 2, 5), bands_names=('red',
                                              'green', 'blue', 'nir'), img_ids=None,
                                              resize=False, normalize=True, standard-
                                              ize=False, metric_fns=[<function nor-
                                              malized_mutual_info_score>, <function
                                              adjusted_rand_score>], mlflow=False,
                                              run_name=None)
```

Get evaluation metrics for given model on L8CCA testset.

#### Parameters

- **model** (Model) – trained model to make predictions.
- **thr** (float) – threshold to be used during evaluation.
- **dpath** (Path) – path to dataset.
- **rpath** (Path) – path to directory 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.
- **bands** (Tuple[int]) – band numbers to load
- **bands\_names** (Tuple[str]) – names of the bands to load. Should have the same number of elements as bands.
- **img\_ids** (Optional[List[str]]) – if given, process only these images.
- **resize** (bool) – whether to resize loaded img to gt.
- **normalize** (bool) – whether to normalize the image.
- **standardize** (bool) – whether to standardize the image.
- **metric\_fns** (List[Callable]) – non-Tensorflow metric functions to run evaluation of the model. Must be of the form func(labels\_true, labels\_pred).
- **mlflow** (bool) – whether to use MLFlow.
- **run\_name** (Optional[str]) – name of the run.

**Return type** Tuple[Dict, List]

**Returns** evaluation metrics and evaluation times for scenes.

```
cloud_detection.evaluate_L8CCA.get_img_pred(path, model, batch_size, bands=(4, 3,
2, 5), bands_names=('red', 'green', 'blue',
'nir'), resize=False, normalize=True, stan-
dardize=False, patch_size=384)
```

Generates prediction for a given image.

#### Parameters

- **path** (Path) – path containing directories with image channels.
- **model** (Model) – trained model to make predictions.
- **batch\_size** (int) – size of generated batches, only one batch is loaded to memory at a time.
- **bands** (Tuple[int]) – band numbers to load

- **bands\_names** (Tuple[str]) – names of the bands to load. Should have the same number of elements as bands.
- **resize** (bool) – whether to resize loaded img to gt.
- **normalize** – whether to normalize the image.
- **standardize** – whether to standardize the image.
- **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.run_evaluation(*, model_hash, thr=0.5,
                                             dpath='datasets/clouds/Landsat-
                                             Cloud-Cover-Assessment-Validation-
                                             Data-Partial', rpath='artifacts/', vids,
                                             batch_size=32, bands, bands_names,
                                             img_ids, resize=False, normalize=False,
                                             standardize=False, mlflow=False,
                                             run_name=None)
```

Load model given model hash and get evaluation metrics on L8CCA testset.

#### Parameters

- **model\_hash** (str) – MLFlow hash of the model to load.
- **thr** (float) – threshold to be used during evaluation.
- **dpath** (str) – path to dataset.
- **rpath** (str) – path to directory 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.
- **bands** (list[int]) – band numbers to load
- **bands\_names** (list[str]) – names of the bands to load. Should have the same number of elements as bands.
- **img\_ids** (list[int]) – if given, process only these images.
- **resize** (bool) – whether to resize loaded img to gt.
- **normalize** (bool) – whether to normalize the image.
- **standardize** (bool) – whether to standardize the image.
- **mlflow** (bool) – whether to use MLFlow.
- **run\_name** (Optional[str]) – name of the run.

## 2.1.6 cloud\_detection.losses module

Custom loss functions and metrics for segmentation.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

**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.

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

\_\_call\_\_ (y\_true, y\_pred)

Calculate 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.

**Parameters** smooth (float) – 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.specificty(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.models module

Keras models for cloud detection.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

`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.

### 2.1.8 cloud\_detection.train\_model module

Train the models for cloud detection.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

`cloud_detection.train_model.train_model(training, valgen, rpath, bn_momentum, learning_rate, stopping_patience, epochs, mlflow)`

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

#### Parameters

- **training** (Sequence) – training set generator.
- **valgen** (Sequence) – validation set generator.
- **rpath** (Path) – path to directory where results and artifacts should be logged.
- **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** Tuple[Model, float]

**Returns** trained model and the best threshold.

## 2.1.9 cloud\_detection.utils module

Various utilities, running tools, img editing etc.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

**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.**build\_paths** (*base\_path*, *patches\_path*, *img\_id*)

Build paths to all files containing image channels.

### Parameters

- **base\_path** (Path) – root path containing directories with image channels.
- **patches\_path** (Path) – path to images patches names to load (if None, all patches will be used).
- **img\_id** (str) – image ID; if specified, load paths for this image only.

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

**Returns** list of dicts containing paths to files with image channels.

cloud\_detection.utils.**combine\_channel\_files** (*red\_file*)

Get paths to 'green', 'blue', 'nir' and 'gt' channel files based on path to the 'red' channel of the given image.

**Parameters** **red\_file** (Path) – path to red channel file.

**Return type** Dict[str, Path]

**Returns** dictionary containing paths to files with each image channel.

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_tf(gt, pred, metric_fns)`  
 Calculates TensorFlow 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.load_38cloud_gt(channel_files)`  
 Load 38-Cloud ground truth mask as array from given files.

**Parameters** **channel\_files** (Dict[str, Path]) – Dict with paths to files containing each channel of an image, must contain key 'gt'.

**Return type** ndarray

**Returns** patch ground truth.

`cloud_detection.utils.load_image_paths(base_path, patches_path=None, split_ratios=[1.0], shuffle=True, img_id=None, seed=42)`  
 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.
- **seed** (int) – random seed for shuffling; relevant only if shuffle=True.

**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.utils.load_l8cca_gt(path)`  
 Load L8CCA Validation Data image ground truth.

**Parameters** **path** (Path) – path containing image gts.

**Return type** ndarray

**Returns** image ground truth.

`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, channel_names=('red', 'green', 'blue', 'nir'), size=None, normalize=True, standardize=False)`

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. Keys should contain channel\_names.
- **channel\_names** (Tuple[str]) – Tuple of channel names to load.
- **size** (Optional[Tuple[int]]) – size of the image. If None, return original size.
- **normalize** (bool) – whether to normalize the image.
- **standardize** (bool) – whether to standardize the image (separately for each band).

**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. RGB 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.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.

`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.

## 2.1.10 cloud\_detection.validate module

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

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

`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.
```

## 2.1.11 cloud\_detection.exp\_pachromatic module

Train and test the models for cloud detection using panchromatic data.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
cloud_detection.exp_pachromatic.main(run_name, train_ids, test_ids, dpath, rpath, vids,  
                                     mlflow, train_size, snow_imgs, batch_size, thr, learn-  
                                     ing_rate, bn_momentum, epochs, stopping_patience,  
                                     bands=(8), bands_names=('panchromatic'))
```

Train and test the U-Net model using L8CCA panchromatic images.

**Parameters**

- **run\_name** (str) – name of the run.
- **train\_ids** (List[Tuple[str]]) – IDs of the training images.
- **test\_ids** (List[Tuple[str]]) – IDs of the testing images.
- **dpath** (str) – path to dataset.
- **rpath** (str) – path to directory where results and artifacts should be logged (randomly named directory will be created to store the results).
- **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).
- **snow\_imgs** (List[str]) – list of 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.
- **bands** (Tuple[int]) – band numbers to load
- **bands\_names** (Tuple[str]) – names of the bands to load. Should have the same number of elements as bands.

## 2.1.12 cloud\_detection.evaluate\_model module

Evaluate the models for cloud detection.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
cloud_detection.evaluate_model.evaluate_model(dataset_name, model, thr, dpath, rpath,
                                              vids, batch_size, img_ids, snow_imgs,
                                              mlflow, gtpath=None, vpath=None,
                                              bands=(4, 3, 2, 5), bands_names=('red',
                                              'green', 'blue', 'nir'), resize=False,
                                              normalize=True, standardize=False)
```

Evaluate given dataset.

### Parameters

- **dataset\_name** (str) – name of the dataset, one of “38Cloud” and “L8CCA”.
- **model** (Model) – trained model to make predictions.
- **thr** (float) – threshold to be used during evaluation.
- **dpath** (Path) – path to dataset.
- **rpath** (Path) – path to directory 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** (List[str]) – if given, process only these images.
- **snow\_imgs** (List[str]) – list of snow images IDs.
- **mlflow** (bool) – whether to use mlflow
- **gtpath** (Optional[Path]) – path to dataset ground truths.
- **vpath** (Optional[Path]) – path to dataset (false color) visualisation images.
- **bands** (Tuple[int]) – band numbers to load
- **bands\_names** (Tuple[str]) – names of the bands to load. Should have the same number of elements as bands.

- **resize** (bool) – whether to resize loaded img to gt.
- **normalize** (bool) – whether to normalize the image (works only if dataset\_name=L8CCA).
- **standardize** (bool) – whether to standardize the image (works only if dataset\_name=L8CCA).

**Return type** Tuple[Dict, List]

**Returns** mean metrics for normal and snow datasets as well as evaluation times for scenes.

### 2.1.13 cloud\_detection.exp\_main module

Train and test the models for cloud detection.

If you plan on using this implementation, please cite our work: @INPROCEEDINGS{Grabowski2021IGARSS, author={Grabowski, Bartosz and Ziaja, Maciej and Kawulok, Michal and Nalepa, Jakub}, booktitle={IGARSS 2021 - 2021 IEEE International Geoscience and Remote Sensing Symposium}, title={Towards Robust Cloud Detection in Satellite Images Using U-Nets}, year={2021}, note={in press}}

```
cloud_detection.exp_main.main(run_name, train_path, C38_path, C38_gtpath, L8CCA_path,
                             vpath, rpath, ppath, vids, mlflow, train_size, train_img, bal-
                             ance_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 for the training set.
- **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.

### 2.1.14 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 .npz data and GT from specified paths

**Parameters**

- **data\_file\_path** (str) – Path to the data .npz file
- **gt\_input\_path** (str) – Path to the GT .npz 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*, *file-*  
*name='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: `.npz`
- **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,          vot-
                                                  ing_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,  
                                                                endmem-  
                                                                bers_path=None,  
                                                                experi-  
                                                                ment_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,          chan-
                                                    nels_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: .npz 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 each 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 sub-folders 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, back-
                                         ground_label=0, channels_idx=0, neigh-
                                         borhood_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, neighbor-
                                   hood_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, neighbor-
                          hood_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 un-mixing 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 each 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**





## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### C

- `cloud_detection`, 23
- `cloud_detection.data_gen`, 5
- `cloud_detection.evaluate_38Cloud`, 7
- `cloud_detection.evaluate_L8CCA`, 9
- `cloud_detection.evaluate_model`, 21
- `cloud_detection.exp_main`, 22
- `cloud_detection.exp_panchromatic`, 20
- `cloud_detection.losses`, 12
- `cloud_detection.models`, 14
- `cloud_detection.scripts.cluster`, 3
- `cloud_detection.scripts.generate_training_patches_paths`, 4
- `cloud_detection.scripts.panchromatic_thresholding`, 4
- `cloud_detection.train_model`, 14
- `cloud_detection.utils`, 15
- `cloud_detection.validate`, 18
- `scripts.quantization`, 44
- `scripts.quantization.freeze_model`, 43
- `scripts.quantization.quantize_runner`, 43
- `scripts.train_model`, 50
- `scripts.unmixing`, 43
- `scripts.unmixing.evaluate_unmixing`, 40
- `scripts.unmixing.train_unmixing`, 41
- `scripts.unmixing.unmixing_experiments_runner`, 42

### m

- `ml_intuition`, 37
- `ml_intuition.data`, 33
- `ml_intuition.data.input_fn`, 23
- `ml_intuition.data.io`, 23
- `ml_intuition.data.loggers`, 26
- `ml_intuition.data.noise`, 26
- `ml_intuition.data.preprocessing`, 29
- `ml_intuition.data.transforms`, 31
- `ml_intuition.models`, 33

### S

- `scripts`, 51
- `scripts.artifacts_reporter`, 44
- `scripts.ensemble`, 40
- `scripts.ensemble.ensemble_runner`, 37
- `scripts.ensemble.evaluate_with_ensemble`, 39
- `scripts.evaluate_model`, 45
- `scripts.experiments_runner`, 46
- `scripts.inference_runner`, 47
- `scripts.predict_with_model`, 49
- `scripts.prepare_data`, 49



## Symbols

`__call__()` (*cloud\_detection.losses.DiceCoefMetric*  
*method*), 12  
`__call__()` (*cloud\_detection.losses.JaccardIndexLoss*  
*method*), 12  
`__call__()` (*cloud\_detection.losses.JaccardIndexMetric*  
*method*), 12  
`__call__()` (*ml\_intuition.data.noise.BaseNoise*  
*method*), 26  
`__call__()` (*ml\_intuition.data.noise.Gaussian*  
*method*), 27  
`__call__()` (*ml\_intuition.data.noise.Impulsive*  
*method*), 27  
`__call__()` (*ml\_intuition.data.noise.Shot* *method*), 28  
`__call__()` (*ml\_intuition.data.transforms.BaseTransform*  
*method*), 31  
`__call__()` (*ml\_intuition.data.transforms.ExtractCentralPixelSpectrumTransform*  
*method*), 31  
`__call__()` (*ml\_intuition.data.transforms.MinMaxNormalize*  
*method*), 31  
`__call__()` (*ml\_intuition.data.transforms.OneHotEncode*  
*method*), 32  
`__call__()` (*ml\_intuition.data.transforms.PerBandMinMaxNormalization*  
*method*), 32  
`__call__()` (*ml\_intuition.data.transforms.RNNSpectralInputTransform*  
*method*), 32  
`__call__()` (*ml\_intuition.data.transforms.SpectralTransform*  
*method*), 33

## A

`align_ground_truth()` (*in* *module*  
*ml\_intuition.data.preprocessing*), 29  
`apply_transformations()` (*in* *module*  
*ml\_intuition.data.transforms*), 33

## B

`BaseNoise` (*class in* *ml\_intuition.data.noise*), 26  
`BaseTransform` (*class in*  
*ml\_intuition.data.transforms*), 31  
`bc()` (*ml\_intuition.data.noise.Params* *property*), 27  
`build_paths()` (*in* *module* *cloud\_detection.utils*), 15

`build_rgb_scene_img()` (*in* *module*  
*cloud\_detection.evaluate\_L8CCA*), 9

## C

`calibrate_2d_input()` (*in* *module*  
*ml\_intuition.data.input\_fn*), 23  
`cloud_detection`  
*module*, 23  
`cloud_detection.data_gen`  
*module*, 5  
`cloud_detection.evaluate_38Cloud`  
*module*, 7  
`cloud_detection.evaluate_L8CCA`  
*module*, 9  
`cloud_detection.evaluate_model`  
*module*, 21  
`cloud_detection.exp_main`  
*module*, 22  
`cloud_detection.exp_panchromatic`  
*module*, 20  
`cloud_detection.losses`  
*module*, 12  
`cloud_detection.models`  
*module*, 14  
`cloud_detection.scripts.cluster`  
*module*, 3  
`cloud_detection.scripts.generate_training_patches_p`  
*module*, 4  
`cloud_detection.scripts.panchromatic_thresholding`  
*module*, 4  
`cloud_detection.train_model`  
*module*, 14  
`cloud_detection.utils`  
*module*, 15  
`cloud_detection.validate`  
*module*, 18  
`collect_artifacts_report()` (*in* *module*  
*scripts.artifacts\_reporter*), 44  
`combine_channel_files()` (*in* *module*  
*cloud\_detection.utils*), 15

## D

`datagen_to_gt_array()` (in module *cloud\_detection.validate*), 18  
*DG\_38Cloud* (class in *cloud\_detection.data\_gen*), 5  
*DG\_L8CCA* (class in *cloud\_detection.data\_gen*), 6  
*DiceCoefMetric* (class in *cloud\_detection.losses*), 12

## E

*Ensemble* (class in *ml\_intuition.models*), 33  
`evaluate()` (in module *scripts.ensemble.evaluate\_with\_ensemble*), 39  
`evaluate()` (in module *scripts.evaluate\_model*), 45  
`evaluate()` (in module *scripts.unmixing.evaluate\_unmixing*), 40  
`evaluate_model()` (in module *cloud\_detection.evaluate\_38Cloud*), 7  
`evaluate_model()` (in module *cloud\_detection.evaluate\_L8CCA*), 9  
`evaluate_model()` (in module *cloud\_detection.evaluate\_model*), 21  
`extract_set()` (in module *ml\_intuition.data.io*), 23  
*ExtractCentralPixelSpectrumTransform* (class in *ml\_intuition.data.transforms*), 31

## F

`f1_score()` (in module *cloud\_detection.losses*), 13  
`false_negatives()` (in module *cloud\_detection.utils*), 15  
`false_positives()` (in module *cloud\_detection.utils*), 15  
`find_best_thr()` (in module *cloud\_detection.validate*), 19  
`find_nearest()` (in module *cloud\_detection.validate*), 19  
`fit()` (*cloud\_detection.scripts.panchromatic\_thresholding.ThresholdingClassifier* static method), 4

## G

*Gaussian* (class in *ml\_intuition.data.noise*), 27  
`generate()` (in module *cloud\_detection.scripts.generate\_training\_patches*), 4  
`generate_models_with_noise()` (*ml\_intuition.models.Ensemble* static method), 34  
`get_all_noise_functions()` (in module *ml\_intuition.data.noise*), 28  
`get_full_scene_img()` (in module *cloud\_detection.evaluate\_38Cloud*), 8  
`get_img_pred()` (in module *cloud\_detection.evaluate\_38Cloud*), 8  
`get_img_pred()` (in module *cloud\_detection.evaluate\_L8CCA*), 10

`get_img_pred_shape()` (in module *cloud\_detection.evaluate\_38Cloud*), 8  
`get_metrics_tf()` (in module *cloud\_detection.utils*), 15  
`get_min_max_vectors()` (*ml\_intuition.data.transforms.PerBandMinMaxNormalization* static method), 32  
`get_model()` (in module *ml\_intuition.models*), 35  
`get_noise_functions()` (in module *ml\_intuition.data.noise*), 28  
`get_padded_cube()` (in module *ml\_intuition.data.preprocessing*), 29  
`get_proba()` (*ml\_intuition.data.noise.BaseNoise* static method), 26

## I

*Impulsive* (class in *ml\_intuition.data.noise*), 27  
`inject_noise()` (in module *ml\_intuition.data.noise*), 28  
`inject_noise_to_weights()` (*ml\_intuition.models.Ensemble* static method), 34

## J

*JaccardIndexLoss* (class in *cloud\_detection.losses*), 12  
*JaccardIndexMetric* (class in *cloud\_detection.losses*), 12

## L

`load_38cloud_gt()` (in module *cloud\_detection.utils*), 16  
`load_image_paths()` (in module *cloud\_detection.utils*), 16  
`load_img_gt()` (in module *cloud\_detection.evaluate\_38Cloud*), 8  
`load_l8cca_gt()` (in module *cloud\_detection.utils*), 16  
`load_metrics()` (in module *ml\_intuition.data.io*), 23  
`load_npy()` (in module *ml\_intuition.data.io*), 24  
`load_pb()` (in module *ml\_intuition.data.io*), 24  
`load_processed_h5()` (in module *ml\_intuition.data.io*), 24  
`load_satellite_h5()` (in module *ml\_intuition.data.io*), 24  
`load_tiff()` (in module *ml\_intuition.data.io*), 24  
`log_dict_as_str_to_mlflow()` (in module *ml\_intuition.data.loggers*), 26  
`log_metrics_to_mlflow()` (in module *ml\_intuition.data.loggers*), 26  
`log_params_to_mlflow()` (in module *ml\_intuition.data.loggers*), 26  
`log_tags_to_mlflow()` (in module *ml\_intuition.data.loggers*), 26

## M

[main\(\) \(in module \*cloud\\_detection.exp\\_main\*\)](#), 22  
[main\(\) \(in module \*cloud\\_detection.exp\\_panchromatic\*\)](#), 20  
[main\(\) \(in module \*scripts.prepare\\_data\*\)](#), 49  
[main\(\) \(in module \*scripts.quantization.freeze\\_model\*\)](#), 43  
[main\\_38Cloud\(\) \(in module \*cloud\\_detection.data\\_gen\*\)](#), 7  
[main\\_L8CCA\(\) \(in module \*cloud\\_detection.data\\_gen\*\)](#), 7  
[make\\_activation\\_hist\(\) \(in module \*cloud\\_detection.validate\*\)](#), 19  
[make\\_paths\(\) \(in module \*cloud\\_detection.utils\*\)](#), 16  
[make\\_precision\\_recall\(\) \(in module \*cloud\\_detection.validate\*\)](#), 19  
[make\\_roc\(\) \(in module \*cloud\\_detection.validate\*\)](#), 19  
[make\\_validation\\_insights\(\) \(in module \*cloud\\_detection.validate\*\)](#), 19  
[mean\(\) \(\*ml\\_intuition.data.noise.Params\* property\)](#), 27  
[MinMaxNormalize \(class in \*ml\\_intuition.data.transforms\*\)](#), 31  
[ml\\_intuition module](#), 37  
[ml\\_intuition.data module](#), 33  
[ml\\_intuition.data.input\\_fn module](#), 23  
[ml\\_intuition.data.io module](#), 23  
[ml\\_intuition.data.loggers module](#), 26  
[ml\\_intuition.data.noise module](#), 26  
[ml\\_intuition.data.preprocessing module](#), 29  
[ml\\_intuition.data.transforms module](#), 31  
[ml\\_intuition.models module](#), 33  
[MLFlowCallback \(class in \*cloud\\_detection.utils\*\)](#), 15  
[model\\_2d\(\) \(in module \*ml\\_intuition.models\*\)](#), 35  
[model\\_3d\\_deep\(\) \(in module \*ml\\_intuition.models\*\)](#), 35  
[model\\_3d\\_mfl\(\) \(in module \*ml\\_intuition.models\*\)](#), 35  
[MODELS \(\*ml\\_intuition.models.Ensemble\* attribute\)](#), 33  
[module \*cloud\\_detection\*](#), 23  
[module \*cloud\\_detection.data\\_gen\*](#), 5  
[module \*cloud\\_detection.evaluate\\_38Cloud\*](#), 7  
[module \*cloud\\_detection.evaluate\\_L8CCA\*](#), 9  
[module \*cloud\\_detection.evaluate\\_model\*](#), 21  
[module \*cloud\\_detection.exp\\_main\*](#), 22  
[module \*cloud\\_detection.exp\\_panchromatic\*](#), 20

[module \*cloud\\_detection.losses\*](#), 12  
[module \*cloud\\_detection.models\*](#), 14  
[module \*cloud\\_detection.scripts.cluster\*](#), 3  
[module \*cloud\\_detection.scripts.generate\\_training\\_patch\*](#), 4  
[module \*cloud\\_detection.scripts.panchromatic\\_threshold\*](#), 4  
[module \*cloud\\_detection.train\\_model\*](#), 14  
[module \*cloud\\_detection.utils\*](#), 15  
[module \*cloud\\_detection.validate\*](#), 18  
[ml\\_intuition](#), 37  
[ml\\_intuition.data](#), 33  
[ml\\_intuition.data.input\\_fn](#), 23  
[ml\\_intuition.data.io](#), 23  
[ml\\_intuition.data.loggers](#), 26  
[ml\\_intuition.data.noise](#), 26  
[ml\\_intuition.data.preprocessing](#), 29  
[ml\\_intuition.data.transforms](#), 31  
[ml\\_intuition.models](#), 33  
[scripts](#), 51  
[scripts.artifacts\\_reporter](#), 44  
[scripts.ensemble](#), 40  
[scripts.ensemble.ensemble\\_runner](#), 37  
[scripts.ensemble.evaluate\\_with\\_ensemble](#), 39  
[scripts.evaluate\\_model](#), 45  
[scripts.experiments\\_runner](#), 46  
[scripts.inference\\_runner](#), 47  
[scripts.predict\\_with\\_model](#), 49  
[scripts.prepare\\_data](#), 49  
[scripts.quantization](#), 44  
[scripts.quantization.freeze\\_model](#), 43  
[scripts.quantization.quantize\\_runner](#), 43  
[scripts.train\\_model](#), 50  
[scripts.unmixing](#), 43  
[scripts.unmixing.evaluate\\_unmixing](#), 40  
[scripts.unmixing.train\\_unmixing](#), 41  
[scripts.unmixing.unmixing\\_experiments\\_runner](#), 42

## N

[normalize\\_labels\(\) \(in module \*ml\\_intuition.data.preprocessing\*\)](#), 29

## O

[on\\_epoch\\_end\(\) \(\*cloud\\_detection.data\\_gen.DG\\_38Cloud\* method\)](#), 6  
[on\\_epoch\\_end\(\) \(\*cloud\\_detection.data\\_gen.DG\\_L8CCA\* method\)](#), 7  
[on\\_epoch\\_end\(\) \(\*cloud\\_detection.utils.MLFlowCallback\* method\)](#), 15

OneHotEncode (class in *ml\_intuition.data.transforms*),  
32  
open\_as\_array() (in module *cloud\_detection.utils*),  
16  
overlay\_mask() (in module *cloud\_detection.utils*),  
17

## P

pa() (*ml\_intuition.data.noise.Params* property), 27  
pad() (in module *cloud\_detection.utils*), 17  
Params (class in *ml\_intuition.data.noise*), 27  
pb() (*ml\_intuition.data.noise.Params* property), 27  
PerBandMinMaxNormalization (class in  
*ml\_intuition.data.transforms*), 32  
pool\_model\_2d() (in module *ml\_intuition.models*),  
35  
precision() (in module *cloud\_detection.losses*), 13  
predict() (*cloud\_detection.scripts.panchromatic\_thresholding.ThresholdingClassifier*  
method), 4  
predict() (in module *scripts.predict\_with\_model*), 49  
predict() (*ml\_intuition.models.Ensemble* method), 34  
predict\_probabilities()  
(*ml\_intuition.models.Ensemble* method),  
34  
pw() (*ml\_intuition.data.noise.Params* property), 28

## R

read\_min\_max() (in module *ml\_intuition.data.io*), 24  
recall() (in module *cloud\_detection.losses*), 13  
remove\_nan\_samples() (in module  
*ml\_intuition.data.preprocessing*), 29  
reshape\_cube\_to\_2d\_samples() (in module  
*ml\_intuition.data.preprocessing*), 29  
reshape\_cube\_to\_3d\_samples() (in module  
*ml\_intuition.data.preprocessing*), 30  
RNNSpectralInputTransform (class in  
*ml\_intuition.data.transforms*), 32  
run() (in module *cloud\_detection.scripts.panchromatic\_thresholding.ThresholdingClassifier*),  
4  
run\_clustering() (in module  
*cloud\_detection.scripts.cluster*), 3  
run\_evaluation() (in module  
*cloud\_detection.evaluate\_38Cloud*), 8  
run\_evaluation() (in module  
*cloud\_detection.evaluate\_L8CCA*), 11  
run\_experiments() (in module  
*scripts.ensemble.ensemble\_runner*), 37  
run\_experiments() (in module  
*scripts.experiments\_runner*), 46  
run\_experiments() (in module  
*scripts.inference\_runner*), 47  
run\_experiments() (in module  
*scripts.quantization.quantize\_runner*), 43

run\_experiments() (in module  
*scripts.unmixing.unmixing\_experiments\_runner*),  
42

## S

save\_confusion\_matrix() (in module  
*ml\_intuition.data.io*), 25  
save\_md5() (in module *ml\_intuition.data.io*), 25  
save\_metrics() (in module *ml\_intuition.data.io*), 25  
save\_ml\_report() (in module *ml\_intuition.data.io*),  
25  
save\_vis() (in module *cloud\_detection.utils*), 17  
scripts  
module, 51  
scripts.artifacts\_reporter  
module, 44  
scripts.ensemble  
scripts.ensemble.ensemble\_runner  
module, 37  
scripts.ensemble.evaluate\_with\_ensemble  
module, 39  
scripts.evaluate\_model  
module, 45  
scripts.experiments\_runner  
module, 46  
scripts.inference\_runner  
module, 47  
scripts.predict\_with\_model  
module, 49  
scripts.prepare\_data  
module, 49  
scripts.quantization  
module, 44  
scripts.quantization.freeze\_model  
module, 43  
scripts.quantization.quantize\_runner  
module, 43  
scripts.train\_model  
module, 50  
scripts.unmixing  
module, 43  
scripts.unmixing.evaluate\_unmixing  
module, 40  
scripts.unmixing.train\_unmixing  
module, 41  
scripts.unmixing.unmixing\_experiments\_runner  
module, 42  
setup\_mlflow() (in module *cloud\_detection.utils*),  
18  
Shot (class in *ml\_intuition.data.noise*), 28  
specificity() (in module *cloud\_detection.losses*),  
13



`spectral_angle_mapper()` (in module `ml_intuition.data.preprocessing`), 30  
`SPECTRAL_DIM` (`ml_intuition.data.transforms.PerBandMinMaxNormalization` attribute), 32  
`SpectralTransform` (class in `ml_intuition.data.transforms`), 33  
`std()` (`ml_intuition.data.noise.Params` property), 28  
`strip_nir()` (in module `cloud_detection.utils`), 18

## T

`ThresholdingClassifier` (class in `cloud_detection.scripts.panchromatic_thresholding`), 4  
`train()` (in module `scripts.train_model`), 50  
`train()` (in module `scripts.unmixing.train_unmixing`), 41  
`train_ensemble_predictor()` (`ml_intuition.models.Ensemble` method), 34  
`train_model()` (in module `cloud_detection.train_model`), 14  
`train_val_test_split()` (in module `ml_intuition.data.preprocessing`), 30  
`true_positives()` (in module `cloud_detection.utils`), 18

## U

`unet()` (in module `cloud_detection.models`), 14  
`unmixing_cube_based_cnn()` (in module `ml_intuition.models`), 36  
`unmixing_cube_based_dcae()` (in module `ml_intuition.models`), 36  
`unmixing_pixel_based_cnn()` (in module `ml_intuition.models`), 36  
`unmixing_pixel_based_dcae()` (in module `ml_intuition.models`), 37  
`unmixing_rnn_supervised()` (in module `ml_intuition.models`), 37  
`unpad()` (in module `cloud_detection.utils`), 18

## V

`validate_demo()` (in module `cloud_detection.validate`), 20  
`vote()` (`ml_intuition.models.Ensemble` method), 34