PrognosAIs

Sebastian van der Voort

Mar 14, 2022

CONTENTS

1 Documentation **1**
- 1.1 Quick start . . . 1
  - 1.1.1 Installation . . . 1
  - 1.1.2 Example experiment . . . 1
- 1.2 Custom network example . . . 2
  - 1.2.1 Implementing your own network . . . 2
    - 1.2.1.1 Basic example: classification network . . . 2
    - 1.2.1.2 Advanced example: multiple inputs/outputs . . . 4
    - 1.2.1.3 Advanced example: non-classification network . . . 5

2 API documentation **7**
- 2.1 PrognosAIs.IO package . . . 7
  - 2.1.1 Submodules . . . 7
  - 2.1.2 PrognosAIs.IO.ConfigLoader module . . . 7
  - 2.1.3 PrognosAIs.IO.Configs module . . . 9
  - 2.1.4 PrognosAIs.IO.DataGenerator module . . . 10
  - 2.1.5 PrognosAIs.IO.LabelParser module . . . 16
  - 2.1.6 PrognosAIs.IO.utils module . . . 18
  - 2.1.7 Module contents . . . 19
- 2.2 PrognosAIs.Model package . . . 19
  - 2.2.1 Subpackages . . . 19
    - 2.2.1.1 PrognosAIs.Model.Architectures package . . . 19
  - 2.2.2 Submodules . . . 29
  - 2.2.3 PrognosAIs.Model.Callbacks module . . . 29
  - 2.2.4 PrognosAIs.Model.Evaluators module . . . 29
  - 2.2.5 PrognosAIs.Model.Losses module . . . 32
  - 2.2.6 PrognosAIs.Model.Metrics module . . . 34
  - 2.2.7 PrognosAIs.Model.Parsers module . . . 38
  - 2.2.8 PrognosAIs.Model.Trainer module . . . 39
  - 2.2.9 Module contents . . . 41
- 2.3 PrognosAIs.Preprocessing package . . . 41
  - 2.3.1 Submodules . . . 41
  - 2.3.2 PrognosAIs.Preprocessing.Preprocessors module . . . 41
  - 2.3.3 PrognosAIs.Preprocessing.Samples module . . . 43
  - 2.3.4 Module contents . . . 49

3 Submodules **51**

4 PrognosAIs.Constants module **53**

5 PrognosAIs.Pipeline module 55

6 Module contents 57

Python Module Index 59

Index 61

CHAPTER

ONE

DOCUMENTATION

1.1 Quick start

This page will show a quick-start of using prognosais, including installation and a simple toy experiment.

1.1.1 Installation

Prognosais can be installed using `pip`. It is recommended to install prognosais in a virtual environment: The following code block creates a virtual environment and installs prognosais in a linux environment.

```
mkdir ~/prognosais && cd ~/prognosais
python3 -m venv env
source env/bin/activate
pip install prognosais
```

1.1.2 Example experiment

We will now set up an example experiment to show how prognosais works and to explain the settings of prognosais. First we set up the experiment by installing the additionaly required packages and obtain the code of the examples: (This assumes that the virtual environment has been set-up as specified under installation)

```
cd ~/prognosais
source env/bin/activate
pip install xlrd
git clone https://github.com/Svdvoort/prognosais_examples
cd prognosais_examples
```

Now, we need to download the data for the example experiment. This data is part of the 'LGG-1p19qDeletion' collection on TCIA More information can be found in the accompanying publication.

We will now download the data to a directory of our choosing:

```
python download_1p19q_data.py
```

You will now be prompted for a directory in which to save the data. Wait untill the data is done downloading and extracting. The script will have prepared the input data and the download data, you can have a look in the download folder you specified.

The script will provide the input folder and label file that need to be specified. Open the `config.yml` file (in the prognosais_examples folder), you can have a look here at the different settings, which are explained more in depth in the file itself. For now we need to change three parameters:

1. input_folder under general, which is set to `/path/to/input/`, needs to be changed to the input folder provided by the download script
2. label_file under preprocessing > labeling which is set to `\path\to\label_file`, needs to be changed to the label file provided by the download script
3. output_folder under general, which is set to `/path/to/output`, needs to be changed to a folder of your choice in which the output will be saved.

If you want to speed-up the pre-processing you can also change the 'max_cpus' setting in preprocessing > general. By default, this is set to 1 which means that only 1 cpu core will be used, increase this if you have multiple cores available.

Once this is done the experiment can simply be run with

```
python example_pipeline.py
```

This will run the pipeline, including the pre-processing of the scans, the training of the model (a ResNet) and the evaluation of the model on the validation and test set. The results will be placed in the folder you specified under 'output_folder', in a subfolder starting with ResNet_18. This folder contains the pre-processed samples, the trained model (including logs from callbacks), and the evaluated results.

1.2 Custom network example

This page will continue the quick-start by showing how to implement and train your own network. It is assumed that you already followed the quick-start and set up the example there.

1.2.1 Implementing your own network

Prognosais was designed to make designing and training your own network as simple as possible.

1.2.1.1 Basic example: classification network

The simplest case is that of a 'classification' network, where samples belong to a discrete class (this can be either a single output label or a segmentation). In this case, only the model itself needs to be implemented.

We start by going to the directory with examples created earlier and creating the virtual environment. Here we will also create file *my_definitions.py* to contain our custom network

```
cd ~/prognosais
source env/bin/activate
cd prognosais_examples/
touch my_definitions.py
```

Now open *my_definitions.py* in your favorite editor and past the following into the file and save it:

```
from tensorflow.keras.layers import Concatenate, Conv3D, Dense, Flatten, ReLU
from tensorflow.keras.models import Model
from PrognosAIs.Model.Architectures.Architecture import␣
↪ClassificationNetworkArchitecture, NetworkArchitecture

class SimpleNetwork_3D(ClassificationNetworkArchitecture):
    # We derive this class from the base class of the classification network
    # The class should be name as followed: {arbitrary_name}_2D for a 2D network or
↪{arbitrary_name}_3D for a 3D network
```

(continues on next page)

(continued from previous page)

```
    # In this way Prognosais will automatically chose the appropriate network based␣
↪on the input dimensions

    def create_model(self):
        # Since we use the ClassificationNetworkArchitecture, we only need to define␣
↪the function create_model
        # This function should construct the model and return it.

        # The inputs are already automatically defined, we can get them from `self`
        # In this case we assume there is only 1 input (for multiple inputs see more␣
↪complicated examples later)
        inputs = self.inputs

        # We will now create a very simple model
        conv_1 = Conv3D(filters=4, kernel_size=(2, 2, 2))(inputs)

        relu_1 = ReLU()(conv_1)

        flatten_1 = Flatten()(relu_1)

        dense_1 = Dense(units=256)(flatten_1)

        # Since we use a ClassificationNetworkArchitecture, the outputs are defined␣
↪already as well
        # In this case by default we get softmax output
        predictions = self.outputs(dense_1)

        # We construct the model and return it

        return Model(inputs=self.inputs, outputs=predictions)
```

Now we need to edit the *config.yml* file in two places:

1. Under *general* add the following: *custom_definitions_file: my_definitions.py*. This will make prognosais load your file with custom definitions
2. Under *model* change *model_name* to *SimpleNetwork*, this will make sure we use our just defined network.

For the *model_name* parameter you never need to add the _2D or _3D part, prognosais will add this automatically based on the dimensions of the input.

The pipeline can now be run again and this new model will be trained:

```
python example_pipeline.py
```

Of course the model will perform very poorly since it is quite simple, but of course you can make the model as complex as you want.

1.2.1.2 Advanced example: multiple inputs/outputs

Creating a network that accepts multiple inputs or outputs is not much more complicated than creating the simple network shown in the previous example. We will expand the previous simple network to deal with multiple inputs and outputs. Once again open the *my_definitions.py* file and add the following code:

```
class NetworkMultiInputMultiOutput_3D(ClassificationNetworkArchitecture):
    def create_model(self):
        # Once again the inputs are automatically created
        # However, since in our toy example data we only have one input and one␣
↪output, we need to override the default settings
        self.inputs = self.make_inputs(self.input_shapes, self.input_data_type,␣
↪squeeze_inputs=False)
        self.outputs = self.make_outputs(self.output_info, self.output_data_type,␣
↪squeeze_outputs=False)
        # By setting squeeze to False, we ensure that even though we do not have␣
↪multiple inputs/outputs, the inputs and outputs will
        # still be created as if there were actually multiple inputs and outputs
        # If you are sure that you always have multiple inputs/outputs you can use␣
↪the self.inputs and self.outputs variables directly
        # Otherwise the above two lines are a safe alternative, making sure your␣
↪model works regardless of the number of inputs/outputs

        # Now the self.inputs variable is actually a dictionary, where the keys are␣
↪the different input names and the values the actual inputs
        # In this case apply a different convolutional filter to each input, and then␣
↪concatenate all the inputs

        input_branches = []
        for i_input in self.inputs.values():
            input_branches.append(Conv3D(filters=4, kernel_size=(2, 2, 2))(i_input))

        # Only concatenate if there is more than 1 input
        if len(input_branches) > 1:
            concat_1 = Concatenate()(input_branches)
        else:
            concat_1 = input_branches[0]

        relu_1 = ReLU()(concat_1)

        flatten_1 = Flatten()(relu_1)

        dense_1 = Dense(units=256)(flatten_1)

        # The output are defined similarly, a dictionary with the keys the names of␣
↪the outputs
        # Thus we can easily create multiple outputs in the following way:
        predictions = []
        for i_output in self.outputs.values():
            predictions.append(i_output(dense_1))

        # If you want to do different things with your outputs you can of course also␣
↪do something like:
        # predictions = []
        # predictions.append(Dense(units=5, activation="softmax", name="output_1")
        # predictions.append(Dense(units=15, activation="relu", name="output_2")
        # Make sure that the name matches the output labels as defined in your label␣
↪file!
```

(continues on next page)

(continued from previous page)

```
        # You can also get the output labels from self.output_info.keys()

        # We construct the model and return it

        return Model(inputs=self.inputs, outputs=predictions)
```

We now need to change the *config.yml* file to train this new network. Simply change *model_name* under *model* to *NetworkMultiInputMultiOutput*, this will make sure we use our just defined network. The model can now be trained:

```
python example_pipeline.py
```

Of course in this example nothing will change compared to the previous example, since our data only has one input and one output.

1.2.1.3 Advanced example: non-classification network

In the above examples we have always used a ClassificationNetworkArchitecture, which makes it easier to implement our own network. However, it is possible to implement any arbitrary network using the more basic NetworkArchitecture, of which we present an example here.

Once again open *my_definitions.py* and add the following:

```
class NonClassificationNetwork_3D(NetworkArchitecture):
    # We have now used the NetworkArchitecture as the base class
    # We use the same model as the first basic example, nothing changed here
    def create_model(self):
        # Since we use the ClassificationNetworkArchitecture, we only need to define␣
↪the function create_model
        # This function should construct the model and return it.

        # We need to load the inputs and outputs, they are not automatically␣
↪generated in this case
        self.inputs = self.make_inputs(self.input_shapes, self.input_data_type)
        self.outputs = self.make_outputs(self.output_info, self.output_data_type)

        # We will now create a very simple model
        conv_1 = Conv3D(filters=4, kernel_size=(2, 2, 2))(self.inputs)

        relu_1 = ReLU()(conv_1)

        flatten_1 = Flatten()(relu_1)

        dense_1 = Dense(units=256)(flatten_1)

        # Since we use a ClassificationNetworkArchitecture, the outputs are defined␣
↪already as well
        # In this case by default we get softmax output
        predictions = self.outputs(dense_1)

        # We construct the model and return it

        return Model(inputs=self.inputs, outputs=predictions)

    # However, we now also need to define a make_outputs function, since we do not␣
↪have default for this for this basic architecture
```

(continues on next page)

(continued from previous page)

```
    @staticmethod
    def make_outputs(
        output_info: dict,
        output_data_type: str,
        activation_type: str = "linear",
        squeeze_outputs: bool = True,
    ) -> dict:
        # The variables output_info and output_date_type are required in any make_
↪outputs function, however apart from that you can
        # create any additional parameters that you want

        # The below code will create a dictionary of outputs (one item for each␣
↪output) and we create a dense layer with one node and linear activation
        # The dtype is float32 but can be adjusted if required for your problem
        outputs = {}
        for i_output_name in output_info.keys():
            outputs[i_output_name] = Dense(
                1, name=i_output_name, activation="linear", dtype="float32",
            )

        # To make it easier for cases where there is only one output we will squeeze␣
↪the output
        # Returning only that output instead of a dict
        if squeeze_outputs and len(outputs) == 1:
            outputs = list(outputs.values())[0]

        return outputs
```

We cannot train this model as the toy example dataset only has discrete data. However, this shows how a model can be implemented that has arbitrary outputs.

CHAPTER

TWO

API DOCUMENTATION

2.1 PrognosAIs.IO package

2.1.1 Submodules

2.1.2 PrognosAIs.IO.ConfigLoader module

class PrognosAIs.IO.ConfigLoader.**ConfigLoader**(*config_file*)

Bases: object

copy_config(*output_folder*, *save_name=None*)

get_N_classes()

get_N_epoch()

get_N_jobs()

get_N_max_patches()

get_batch_size()

get_cache_in_memory()

get_callback_settings()

get_center_patch_around_mask()

get_class_weights()

get_cluster_setting()

get_cluster_type()

get_combine_patch_predictions()

get_config_file()

get_copy_files()

get_custom_definitions_file()

get_data_augmentation()

get_data_augmentation_factor()

get_data_augmentation_settings()

get_data_folder()

get_dataset_distribution()

get_do_augmentation()

get_dtype()

get_evaluate_metrics()

get_evaluate_train_set()

get_evaluation_mask_labels()

get_evaluation_metric_settings()

get_extra_input_file()

get_filter_missing()

get_float16_epsilon()

get_float_policy()

get_fsl_reorient_bin()

get_fsl_val_bin()

get_gpu_workers()

get_image_size()

get_input_folder()

get_keep_rejected_patches()

get_label_combination_type()

get_label_file()

get_loss_settings()

get_loss_weights()

get_make_one_hot()

get_make_patches()

get_mask_file()

get_mask_keyword()

get_max_steps_per_epoch()

get_metric_settings()

get_min_patch_voxels()

get_model_file()

get_model_name()

get_model_settings()

get_multi_channels_patches()

get_optimizer_settings()

get_output_folder()

get_patch_predictions()

get_patch_size()

get_preprocessings_settings()

get_processed_samples_folder()

get_random_state()

get_reject_patches()

get_resample_images()

get_resample_size()

get_rescale_mask_intensity()

get_resume_training_from_model()

get_save_name()

get_shuffle()

get_shuffle_evaluation()

get_shuffle_val()

get_size_string()

get_specific_output_folder()

get_stratify_index()

get_test_data_folder()

get_test_label_file()

get_test_model_file()

get_training_multi_processing()

get_use_class_weights()

get_use_class_weights_in_losses()

get_use_labels_from_rejection()

get_use_mask_as_channel()

get_use_mask_as_label()

get_write_predictions()

2.1.3 PrognosAIs.IO.Configs module

class PrognosAIs.IO.Configs.**bias_field_correcting_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

property mask

property mask_file

class PrognosAIs.IO.Configs.**config**(*config_settings: Optional[dict]*)
Bases: object

static get_step_type(*config: Optional[dict]*) → Tuple[bool, bool, dict]

class PrognosAIs.IO.Configs.**general_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

class PrognosAIs.IO.Configs.**labeling_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

class PrognosAIs.IO.Configs.**masking_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

property mask

property mask_file

class PrognosAIs.IO.Configs.**multi_dimension_extracting_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

class PrognosAIs.IO.Configs.**normalizing_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

property mask

property mask_file

class PrognosAIs.IO.Configs.**patching_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

property patch_size

class PrognosAIs.IO.Configs.**rejecting_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

property mask

property mask_file

class PrognosAIs.IO.Configs.**resampling_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

class PrognosAIs.IO.Configs.**saving_config**(*config_settings: dict*)
Bases: *PrognosAIs.IO.Configs.config*

2.1.4 PrognosAIs.IO.DataGenerator module

class PrognosAIs.IO.DataGenerator.**Augmentor**(*example_sample: tensorflow.python.framework.ops.Tensor, brightness_probability: float = 0, brightness_delta: float = 0, contrast_probability: float = 0, contrast_min_factor: float = 1, contrast_max_factor: float = 1, flip_probability: float = 0, to_flip_axis: Union[int, list] = 0, crop_probability: float = 0, crop_size: list = None, rotate_probability: float = 0, max_rotate_angle: float = 0, to_rotate_axis: Union[int, list] = 0*)
Bases: object

__init__(*example_sample: tensorflow.python.framework.ops.Tensor, brightness_probability: float = 0, brightness_delta: float = 0, contrast_probability: float = 0, contrast_min_factor: float = 1, contrast_max_factor: float = 1, flip_probability: float = 0, to_flip_axis: Union[int, list] = 0, crop_probability: float = 0, crop_size: list = None, rotate_probability: float = 0, max_rotate_angle: float = 0, to_rotate_axis: Union[int, list] = 0*) → None
Augmentor to randomly augment the features of a sample.

Parameters

- **example_sample** (*tf.Tensor*) – Example sample from which settings for augmentation will be derived

- **brightness_probability** (*float, optional*) – Probability of augmenting brightness. Defaults to 0.
- **brightness_delta** (*float, optional*) – Brightness will be adjusted with value from -delta to delta. Defaults to 0.
- **contrast_probability** (*float, optional*) – Probability of augmenting contrast. Defaults to 0.
- **contrast_min_factor** (*float, optional*) – Minimum contrast adjustment factor. Defaults to 1.
- **contrast_max_factor** (*float, optional*) – Maximum contrast adjustment factor. Defaults to 1.
- **flip_probability** (*float, optional*) – Probability of a random flip. Defaults to 0.
- **to_flip_axis** (*Union[int, list], optional*) – Axis to flip the feature over. Defaults to 0.
- **crop_probability** (*float, optional*) – Probability of cropping the feature. Defaults to 0.
- **crop_size** (*list, optional*) – Size to crop the feature to. Defaults to None.

apply_augmentation(*augmentation_probability: float*, *seed: tensorflow.python.framework.ops.Tensor = None*) → bool

Whether the the augmentation step should be applied based on the probability.

Parameters

- **augmentation_probability** (*float*) – The probability with which the step should be applied
- **seed** (*tf.Tensor*) – Seed to make operation repeatable. Defaults to None.

Returns *bool* – Whether the step should be applied

augment_sample(*sample: tensorflow.python.framework.ops.Tensor*, *seed=None*, *is_mask=False*) → tensorflow.python.framework.ops.Tensor

Apply random augmentations to the sample based on the config.

Parameters **sample** (*tf.Tensor*) – sample to be augmented

Returns *tf.Tensor* – augmented sample

get_seed() → tensorflow.python.framework.ops.Tensor

Get a random seed that can be used to make other operation repeatable.

Returns *tf.Tensor* – The seed

pad_to_original_size(*sample: tensorflow.python.framework.ops.Tensor*) → tensorflow.python.framework.ops.Tensor

Pad back a (potentially) augmented sample to its original size.

Parameters **sample** (*tf.Tensor*) – The sample to pad

Returns *tf.Tensor* – The padded sample with the same size as before any augmentation steps

random_brightness(*sample: tensorflow.python.framework.ops.Tensor*, *seed: tensorflow.python.framework.ops.Tensor = None*) → tensorflow.python.framework.ops.Tensor

Randomly adjusts the brightness of a sample.

Brightness is adjusted by a contact factor over the whole image, drawn from a distribution between -delta and delta as set during the initialization of the augmentator.

Parameters

- **sample** (*tf.Tensor*) – Sample for which to adjust brightness.

- **seed** (*tf.Tensor*) – Seed to make operation repeatable. Defaults to None.

Returns *tf.Tensor* – The augmented sample.

random_contrast(*sample: tensorflow.python.framework.ops.Tensor*, *seed: tensorflow.python.framework.ops.Tensor = None*) → tensorflow.python.framework.ops.Tensor

Randomly adjust the contrast of a sample.

The contrast is adjusted by keeping the mean of the sample the same as for the original sample, and squeezing or expending the distribution of the intensities around the mean. The amount of squeezing or expanding is randomly drawn from the minimum and maximum contrast set during initialization.

Parameters

- **sample** (*tf.Tensor*) – Sample for which to adjust contrast
- **seed** (*tf.Tensor*) – Seed to make operation repeatable. Defaults to None.

Returns *tf.Tensor* – The augmented sample

random_cropping(*sample: tensorflow.python.framework.ops.Tensor*, *seed: tensorflow.python.framework.ops.Tensor = None*) → tensorflow.python.framework.ops.Tensor

Randomly crop a part of the sample.

The crop will have the size of the crop size defined upon initialization of the augmentator. The crop will happen for all channels in the same way, but will not crop out channels. The location of the crop will be randomly drawn from throughout the whole image.

Parameters

- **sample** (*tf.Tensor*) – The sample to be cropped
- **seed** (*tf.Tensor*) – Seed to make operation repeatable. Defaults to None.

Returns *tf.Tensor* – The augmented sample

random_flipping(*sample: tensorflow.python.framework.ops.Tensor*, *seed: tensorflow.python.framework.ops.Tensor = None*) → tensorflow.python.framework.ops.Tensor

Randomly flip the sample along one or multiple axis.

Parameters

- **sample** (*tf.Tensor*) – Sample for which to apply flipping
- **seed** (*tf.Tensor*) – Seed to make operation repeatable. Defaults to None.

Returns *tf.Tensor* – The augmented sample

random_rotate(*feature: tensorflow.python.framework.ops.Tensor*, *seed: tensorflow.python.framework.ops.Tensor = None*, *interpolation_order: int = 3*) → tensorflow.python.framework.ops.Tensor

class ProgrosAIs.IO.DataGenerator.**HDF5Generator**(*root_folder: str*, *batch_size: int = 16*, *shuffle: bool = False*, *max_steps: int = - 1*, *drop_batch_remainder: bool = True*, *labels_only: bool = False*)

Bases: object

__init__(*root_folder: str*, *batch_size: int = 16*, *shuffle: bool = False*, *max_steps: int = - 1*, *drop_batch_remainder: bool = True*, *labels_only: bool = False*) → None

Generate data from HDF5 files to be used in a TensorFlow pipeline.

This generator loads sample data from HDF5 files, and does this efficiently making us of TensorFlow dataset functions. The inputs and outputs are dict, which allows for easy us in a multi-input and/or multi-output model

Parameters

- **root_folder** (*str*) – Folder in which the HDF5 files are stored
- **batch_size** (*int, optional*) – Batch size of the generator. Defaults to 16.
- **shuffle** (*bool, optional*) – Whether datset should be shuffled. Defaults to False.
- **data_augmentation** (*bool, optional*) – Whether data augmentation should be applied. Defaults to False.
- **augmentation_factor** (*int, optional*) – Number of times dataset should be repeated for augmentation. Defaults to 5.
- **augmentation_settings** (*dict, optional*) – Setting for the data augmenation. Defaults to None.
- **max_steps** (*int, optional*) – Maximum number of (iteration) steps to provide. Defaults to -1, in which case all samples are provied.
- **drop_batch_remainder** (*bool, optional*) – Whether to drop the remainder of the batch if it does not fit perfectly. Defaults to True.
- **labels_only** (*bool, optional*) – Whether to only provide labels. Defaults to False.
- **feature_index** (*str, optional*) – Name of the feature group in the HDF5 file. Defaults to "sample".
- **label_index** (*str, optional*) – Name of the label group in the HDF5 file. Defaults to "label".

_get_all_dataset_attributes (*h5py_object: Union[h5py._hl.files.File, h5py._hl.dataset.Dataset, h5py._hl.group.Group]*) → dict

Run through al groups and dataset to get the attributes.

Parameters **h5py_object** (*Union[h5py.File, h5py.Dataset, h5py.Group]*) – Object for which to return the attributes

Returns *dict* – Mapping between feature/label name and its attributes

_get_dataset_names (*h5py_object: Union[h5py._hl.files.File, h5py._hl.dataset.Dataset, h5py._hl.group.Group]*) → list

Run through all groups and dataset to get the names.

Parameters **h5py_object** (*Union[h5py.File, h5py.Dataset, h5py.Group]*) – Object for which to return the dataset names

Returns *list* – Dataset names in object

apply_augmentation (*features: dict*, *labels: dict*) → Tuple[dict, dict]

feature_loader (*sample_location: tensorflow.python.framework.ops.Tensor*) → dict

Load the features from a hdf5 sample file.

This loader only loads the labels, instead of the features and labels as done by features_and_labels_loader

Parameters **sample_location** (*tf.Tensor*) – Location of the sample file

Returns *dict* – Features loaded from the sample file

features_and_labels_loader (*sample_location: tensorflow.python.framework.ops.Tensor*) → Tuple[dict, dict, tensorflow.python.framework.ops.Tensor]

Load the features and labels from a hdf5 file to be used in a TensorFlow dataset pipeline.

This loader loads the features and labels from a hdf5 file using TensorFlowIO. The outputs are therefor directly cast to tensor and can be used in a TensorFlow graph. All features and labels from the file are loaded, and a dict is returned mapping the name of each feature and label to its respective value

Parameters **sample_location** (*tf.Tensor*) – Location of the sample file

Returns

Tuple[dict, dict] –

The features (first output) and labels (second output) loaded from the sample.

fits_in_memory(*used_memory: int = 0*)

get_all_dataset_attributes(*sample_file: str = None*) → dict
Get the attributes of the features and labels stored in the file.

Returns *dict* – Mapping of the feature/label name to its attributes

get_dataset_attribute(*dataset_name: str*, *attribute_name: str*) → Any
Get the attribute of a specific dataset

Parameters

- **dataset_name** (*str*) – Name of dataset for which to get the attribute
- **attribute_name** (*str*) – Name of attribute to get

Returns *Any* – The value of the attribute

get_dataset_names() → list
Get the names of all datasets in the sample.

Returns *list* – Dataset names in the sample

get_feature_attribute(*attribute_name: str*) → dict
Get a specific attribute for all features.

Parameters **attribute_name** (*str*) – Name of attribute to get

Returns *dict* – Mapping between feature names and the attribute value

get_feature_dimensionality() → dict
Get the dimensionality of each feature.

Returns *dict* – Dimensionality of each feature

get_feature_metadata() → dict
Get all metadata of all features.

Returns *dict* – The metadata of all features

get_feature_metadata_from_sample(*sample_location: str*) → dict
Get the feature metadata of a specific sample.

Parameters **sample_location** (*str*) – The file location of the sample

Returns *dict* – The feature metadata of the sample

get_feature_shape() → dict
Get the shape of each feature.

Returns *dict* – Shape of each feature

get_feature_size() → dict
Get the size of each feature.

The size only of the feature does not take into account the number of channels and only represents the size of an individual channel of the feature.

Returns *dict* – Size of each feature

get_label_attribute(*attribute_name: str*) → dict
Get a specific attribute for all labels.

Parameters **attribute_name** (*str*) – Name of attribute to get

Returns *dict* – Mapping between label names and the attribute value

get_labels_are_one_hot() → dict
Get whether labels are one-hot encoded.

Returns *dict* – One-hot encoding status of each label

get_number_of_channels() → dict
Get the number of feature channels.

Returns *dict* – Number of channels for each feature

get_number_of_classes() → dict
Get the number of output classes.

Returns *dict* – Number of output classes for each label

get_numpy_iterator() → numpy.nditer
Construct a numpy iterator instead of TensorFlow dataset.

The numpy iterator will provide exactly the same data as the TensorFlow dataset. However, it might be easier to inspect the data when using a numpy iterator instead of a TensorFlow dataset

Returns *np.nditer* – The dataset

get_spec() → dict
Get the TensorSpec for all input features.

Returns *dict* – Maps the name of each input feature to the TensorSpec of the input.

get_tf_dataset(*num_parallel_calls: int = -1*) → tensorflow.python.data.ops.dataset_ops.DatasetV2
Construct a TensorFlow dataset.

The dataset is constructed based on the settings supplied to the DataGenerator. The dataset can then directly be used to train or evaluate a TensorFlow model

Parameters **num_parallel_calls** (*int*) – Number of parallel process to use. Defaults to tf.data.experimental.AUTOTUNE.

Returns *tf.data.Dataset* – The constructed dataset

label_loader(*sample_location: tensorflow.python.framework.ops.Tensor*) → dict
Load the labels from a hdf5 sample file.

This loader only loads the labels, instead of the features and labels as done by features_and_labels_loader

Parameters **sample_location** (*tf.Tensor*) – Location of the sample file

Returns *dict* – Labels loaded from the sample file

load_features(*loaded_hdf5: tensorflow_io.core.python.ops.io_tensor.IOTensor*) → dict
Load the features from a HDF5 tensor.

Parameters **loaded_hdf5** (*tfio.IOTensor*) – Tensor from which to load features

Returns *dict* – Mapping between feature names and features

load_labels (*loaded_hdf5: tensorflow_io.core.python.ops.io_tensor.IOTensor*) → dict
Load the labels from a HDF5 tensor.

Parameters **loaded_hdf5** (*tfio.IOTensor*) – Tensor from which to load labels

Returns *dict* – Mapping between label names and labels

setup_augmentation (*augmentation_factor: int = 1*, *augmentation_settings: dict = {}*) → None
Set up data augmentation in the generator.

Parameters

- **augmentation_factor** (*int*) – Repeat dataset this many times in augmentation. Defaults to 1.
- **augmentation_settings** (*dict*) – Setting to parse to augmentation instance. Defaults to {}.

setup_caching (*cache_in_memory: Union[bool, str] = 'AUTO'*, *used_memory: int = 0*) → None
Set up caching of the dataset in RAM.

Parameters

- **cache_in_memory** (*Union[bool, str]*) – Whether dataset should be cached in memory. Defaults to PrognosAIs.Constants.AUTO, in which case the dataset will be cached in memory if it fits, otherwise it will not be cached
- **used_memory** (*int*) – Amount of RAM (in bytes) that is already being used. Defaults to 0.

Raises **ValueError** – If an unknown cache setting is requested

setup_caching_shuffling_steps (*dataset: tensorflow.python.data.ops.dataset_ops.DatasetV2*) → tensorflow.python.data.ops.dataset_ops.DatasetV2
Set-up caching, shuffling and the iteration step in the dataset pipeline.

This function helps to ensure that caching, shuffling and step limiting is done properly and efficiently, no matter where in the dataset pipeline it is included.

Parameters **dataset** (*tf.data.Dataset*) – Datset for which to include the steps

Returns *tf.data.Dataset* – Datset with caching, shuffling and iteration steps included

setup_sharding (*n_workers: int*, *worker_index: int*) → None
Shard the dataset according to the number of workers and worker index

Parameters

- **n_workers** (*int*) – number of workers
- **worker_index** (*int*) – worker index

2.1.5 PrognosAIs.IO.LabelParser module

class PrognosAIs.IO.LabelParser.**LabelLoader** (*label_file: str*, *filter_missing: bool = False*, *missing_value: int = - 1*, *make_one_hot: bool = False*, *new_root_path: str = None*)
Bases: object

__init__ (*label_file: str*, *filter_missing: bool = False*, *missing_value: int = - 1*, *make_one_hot: bool = False*, *new_root_path: str = None*) → None
Create a label loader, that can load the image paths and labels from a text file to be used for a data generator

Parameters

- **label_file** – The label file from which to read the labels

- **filter_missing** – Whether missing values should be masked when generating one hot labels and class weights
- **missing_value** – If filter_missing is True, this value is used to mask
- **make_one_hot** – Whether labels should be transformed to one hot labels
- **new_root_path** – If you want to move the files, this will be the new root path

encode_labels_one_hot() → None
Encode sample labels as one hot

Parameters None

Returns None

get_class_weights(*json_serializable=False*) → dict
Get class weights for unbalanced labels

Parameters None

Returns

Scaled_weights –

the weights for each class of each label category, scaled such that the total weights*number of samples of each class approximates the total number of samples

get_data() → dict
Get all data from the label file

Parameters None

Returns *data* – Dictionary mapping each sample to each label

get_label_categories() → list
Get categories of labels

Parameters None

Returns *label_categories* – Category names

get_label_category_type(*category_name: str*) → type
Get the type of a label of a specific category/class

Parameters **category_name** – Name of the category/class to get type of

Returns *type* – Type of the labels of the category

get_label_from_sample(*sample: str*) → dict
Get label from a sample

Parameters **sample** – The sample from which to get the label

Returns *label* – Label of the sample

get_labels() → list
Get all labels of all samples

Parameters None

Returns *labels* – List of labels

get_labels_from_category(*category_name: str*) → list
Get labels of a specific category/class

Parameters **category_name** – Name of the category/class to get

Returns *list* – Labels of the category

get_number_of_classes() → dict
Get number of classes for all categories

Parameters **None**

Returns *number_of_classes* – The number of classes for each category

get_number_of_classes_from_category(*category_name: str*) → int
Get number of classes for a label category

Parameters **category_name** – Category to get number of classes for

Returns *number_of_classes* – The number of classes for the category

get_number_of_samples() → int
Get number of samples

Parameters **None**

Returns *number_of_samples* – The number of samples

get_original_label_category_type(*category_name: str*) → type
Get the original type of a label of a specific category/class

Parameters **category_name** – Name of the category/class to get type of

Returns *type* – Type of the labels of the category

get_original_labels_from_category(*category_name: str*) → list
Get original labels of a specific category/class

Parameters **category_name** – Name of the category/class to get

Returns *list* – Original labels of the category

get_samples() → list
Get all labels of all samples

Parameters **None**

Returns *samples* – List of samples

replace_root_path() → None
Replace the root path of the sample files in case they have been moved to a different a different directory.

Parameters **new_root_path** – Path in which the files are now located

Returns None

2.1.6 PrognosAIs.IO.utils module

PrognosAIs.IO.utils.**copy_directory**(*original_directory*, *out_directory*)

PrognosAIs.IO.utils.**create_directory**(*file_path*, *exist_ok=True*)

PrognosAIs.IO.utils.**delete_directory**(*file_path*)

PrognosAIs.IO.utils.**find_files_with_extension**(*file_path*, *file_extension*)

PrognosAIs.IO.utils.**get_available_ram**(*used_memory: int = 0*) → int
Get the available RAM in bytes.

Returns *int* – available in RAM in bytes

PrognosAIs.IO.utils.**get_cpu_devices**() → list

PrognosAIs.IO.utils.**get_dir_size**(*root_dir*)
Returns total size of all files in dir (and subdirs)

PrognosAIs.IO.utils.**get_file_name**(*file_path*, *file_extension*)

PrognosAIs.IO.utils.**get_file_name_from_full_path**(*file_path*)

PrognosAIs.IO.utils.**get_file_path**(*file_path*)

PrognosAIs.IO.utils.**get_gpu_compute_capability**(*gpu: tensorflow.python.eager.context.PhysicalDevice*) → tuple

PrognosAIs.IO.utils.**get_gpu_devices**() → list

PrognosAIs.IO.utils.**get_number_of_cpus**()

PrognosAIs.IO.utils.**get_number_of_gpu_devices**() → int

PrognosAIs.IO.utils.**get_number_of_slurm_nodes**() → int

PrognosAIs.IO.utils.**get_parent_directory**(*file_path*)

PrognosAIs.IO.utils.**get_root_name**(*file_path*)

PrognosAIs.IO.utils.**get_subdirectories**(*root_dir: str*) → list

PrognosAIs.IO.utils.**gpu_supports_float16**(*gpu: tensorflow.python.eager.context.PhysicalDevice*) → bool

PrognosAIs.IO.utils.**gpu_supports_mixed_precision**(*gpu: tensorflow.python.eager.context.PhysicalDevice*) → bool

PrognosAIs.IO.utils.**load_module_from_file**(*module_path*)

PrognosAIs.IO.utils.**normalize_path**(*path*)

PrognosAIs.IO.utils.**setup_logger**()

2.1.7 Module contents

2.2 PrognosAIs.Model package

2.2.1 Subpackages

2.2.1.1 PrognosAIs.Model.Architectures package

Submodules

PrognosAIs.Model.Architectures.AlexNet module

class PrognosAIs.Model.Architectures.AlexNet.**AlexNet_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)
Bases: *PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArchitecture*

create_model()
Here the code to create the actual model

padding_type = 'valid'

class PrognosAIs.Model.Architectures.AlexNet.**AlexNet_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)
Bases: *PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArchitecture*

create_model()
Here the code to create the actual model

padding_type = 'valid'

PrognosAIs.Model.Architectures.Architecture module

class PrognosAIs.Model.Architectures.Architecture.**ClassificationNetworkArchitecture**(*input_sha dict*, *output_info: dict*, *input_data_ output_data_ model_co*
Bases: *PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture*

static make_outputs(*output_info: dict*, *output_data_type: str*, *activation_type: str = 'softmax'*, *squeeze_outputs: bool = True*) → dict
Make the outputs

class PrognosAIs.Model.Architectures.Architecture.**NetworkArchitecture**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config: dict = {}*)
Bases: abc.ABC

static check_minimum_input_size(*input_layer: tensorflow.python.keras.engine.input_layer.Input*, *minimum_input_size: numpy.ndarray*)

abstract create_model()
Here the code to create the actual model

static get_corrected_stride_size(*layer: <module 'tensorflow.keras.layers' from '/home/docs/checkouts/readthedocs.org/user_builds/prognosais/envs/stable/lib/python3.7/site-packages/tensorflow/keras/layers/__init__.py'>*, *stride_size: list*, *conv_size: list*)

Ensure that the stride is never bigger than the actual input In this way any network can keep working, indepedent of size

make_dropout_layer(*layer*)

make_inputs(*input_shapes: dict*, *input_dtype: str*, *squeeze_inputs: bool = True*) → Union[dict, tensorflow.python.keras.engine.input_layer.Input]

abstract make_outputs(*output_info: dict*, *output_data_type: str*) → <module 'tensorflow.keras.layers' from '/home/docs/checkouts/readthedocs.org/user_builds/prognosais/envs/stable/lib/python3.7/site-packages/tensorflow/keras/layers/__init__.py'>

Make the outputs

PrognosAIs.Model.Architectures.DDSNet module

class PrognosAIs.Model.Architectures.DDSNet.**DDSNet**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArchitecture*

get_DDS_block(*layer*, *N_filters*)

init_dimensionality(*N_dimension*)

class PrognosAIs.Model.Architectures.DDSNet.**DDSNet_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DDSNet.DDSNet*

create_model()

Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.DDSNet.**DDSNet_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DDSNet.DDSNet*

create_model()

Here the code to create the actual model

dims = 3

PrognosAIs.Model.Architectures.DenseNet module

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArchitecture*

get_dense_block(*layer*, *N_filters*, *N_conv_layers*)

get_dense_stem(*layer*, *N_filters*)

get_transition_block(*layer*, *N_filters*, *theta*)

init_dimensionality(*N_dimension*)

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_121_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()

Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_121_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()

Here the code to create the actual model

dims = 3

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_169_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()
Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_169_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()
Here the code to create the actual model

dims = 3

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_201_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()
Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_201_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()
Here the code to create the actual model

dims = 3

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_264_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()
Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.DenseNet.**DenseNet_264_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.DenseNet.DenseNet*

GROWTH_RATE = 32

INITIAL_FILTERS = 64

THETA = 0.5

create_model()
Here the code to create the actual model

dims = 3

PrognosAIs.Model.Architectures.InceptionNet module

class PrognosAIs.Model.Architectures.InceptionNet.**InceptionNet_InceptionResNetV2_2D**(*input_sha*
dict,
out-
put_info:
dict,
in-
*put_data_*
out-
*put_data_*
model_co

Bases: *PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet*

create_model()
Here the code to create the actual model

class PrognosAIs.Model.Architectures.InceptionNet.**InceptionNet_InceptionResNetV2_3D**(*input_sha*
dict,
out-
put_info:
dict,
in-
*put_data_*
out-
*put_data_*
model_co

Bases: *PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet*

create_model()
Here the code to create the actual model

class PrognosAIs.Model.Architectures.InceptionNet.**InceptionResNet**(*input_shapes:*
dict, *out-*
put_info:
dict, *in-*
put_data_type='float32',
out-
put_data_type='float32',
model_config={})

Bases: *PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArchitecture*

get_inception_resnet_A(*layer*)

get_inception_resnet_B(*layer*)

get_inception_resnet_C(*layer*)

get_inception_resnet_reduction_A(*layer*)

get_inception_resnet_reduction_B(*layer*)

get_inception_stem(*layer*)

init_dimensionality(*N_dimension*)

PrognosAIs.Model.Architectures.ResNet module

class PrognosAIs.Model.Architectures.ResNet.**ResNet**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArchitecture*

get_residual_conv_block(*layer: <module 'tensorflow.keras.layers' from '/home/docs/checkouts/readthedocs.org/user_builds/prognosais/envs/stable/lib/python3.7/site-packages/tensorflow/keras/layers/__init__.py'>*, *N_filters: int*, *kernel_size: list*)

get_residual_identity_block(*layer*, *N_filters*, *kernel_size*)

class PrognosAIs.Model.Architectures.ResNet.**ResNet_18_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.ResNet.ResNet*

create_model()

Here the code to create the actual model

class PrognosAIs.Model.Architectures.ResNet.**ResNet_18_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.ResNet.ResNet*

create_model()

Here the code to create the actual model

class PrognosAIs.Model.Architectures.ResNet.**ResNet_18_multioutput_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.ResNet.ResNet*

create_model()

Here the code to create the actual model

class PrognosAIs.Model.Architectures.ResNet.**ResNet_34_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.ResNet.ResNet*

create_model()
Here the code to create the actual model

class PrognosAIs.Model.Architectures.ResNet.**ResNet_34_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)
Bases: *PrognosAIs.Model.Architectures.ResNet.ResNet*

create_model()
Here the code to create the actual model

PrognosAIs.Model.Architectures.UNet module

class PrognosAIs.Model.Architectures.UNet.**UNet_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config: dict = {}*)
Bases: *PrognosAIs.Model.Architectures.UNet.Unet*

create_model()
Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.UNet.**UNet_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config: dict = {}*)
Bases: *PrognosAIs.Model.Architectures.UNet.Unet*

create_model()
Here the code to create the actual model

dims = 3

class PrognosAIs.Model.Architectures.UNet.**Unet**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config: dict = {}*)
Bases: *PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture*

get_conv_block(*layer*, *N_filters*, *kernel_size=3*, *activation='relu'*, *kernel_regularizer=None*)

get_cropping_block(*conv_layer*, *upsampling_layer*)

get_depth()

get_number_of_filters()

get_padding_block(*layer*)

get_pool_block(*layer*)

get_upsampling_block(*layer*, *N_filters*, *activation='relu'*, *kernel_regularizer=None*)

init_dimensionality(*N_dimension*)

make_outputs(*output_info: dict*, *output_data_type: str*, *activation_type: str = 'softmax'*)
Make the outputs

PrognosAIs.Model.Architectures.VGG module

class PrognosAIs.Model.Architectures.VGG.**VGG**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArchitecture*

get_VGG_block(*layer*, *N_filters*, *N_conv_layer*)

init_dimensionality(*N_dimension*)

class PrognosAIs.Model.Architectures.VGG.**VGG_16_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.VGG.VGG*

create_model()
Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.VGG.**VGG_16_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.VGG.VGG*

create_model()
Here the code to create the actual model

dims = 3

class PrognosAIs.Model.Architectures.VGG.**VGG_19_2D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.VGG.VGG*

create_model()
Here the code to create the actual model

dims = 2

class PrognosAIs.Model.Architectures.VGG.**VGG_19_3D**(*input_shapes: dict*, *output_info: dict*, *input_data_type='float32'*, *output_data_type='float32'*, *model_config={}*)

Bases: *PrognosAIs.Model.Architectures.VGG.VGG*

create_model()
Here the code to create the actual model

dims = 3

Module contents

2.2.2 Submodules

2.2.3 PrognosAIs.Model.Callbacks module

class PrognosAIs.Model.Callbacks.**ConcordanceIndex**(*validation_generator*)
Bases: tensorflow.python.keras.callbacks.Callback

A custom callback function to evaluate the concordance index on the whole validation set

on_epoch_end(*epoch*, *logs=None*)
Called at the end of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Parameters

- **epoch** – Integer, index of epoch.
- **logs** – Dict, metric results for this training epoch, and for the validation epoch if validation is performed. Validation result keys are prefixed with *val_*.

class PrognosAIs.Model.Callbacks.**Timer**
Bases: tensorflow.python.keras.callbacks.Callback

A custom callback function to evaluate the elapsed time of training

on_epoch_end(*epoch*, *logs=None*)
Called at the end of an epoch.

Subclasses should override for any actions to run. This function should only be called during TRAIN mode.

Parameters

- **epoch** – Integer, index of epoch.
- **logs** – Dict, metric results for this training epoch, and for the validation epoch if validation is performed. Validation result keys are prefixed with *val_*.

PrognosAIs.Model.Callbacks.**calculate_concordance_index**(*y_true*, *y_pred*)
This function determine the concordance index for two numpy arrays

y_true contains a label to indicate whether events occurred, and time to events (or time to right censored data if no event occurred)

y_pred is beta*x in the cox model

2.2.4 PrognosAIs.Model.Evaluators module

class PrognosAIs.Model.Evaluators.**Evaluator**(*model_file*, *data_folder*, *config_file*, *output_folder*)
Bases: object

_combine_config_and_model_metrics(*model_metrics: dict*, *config_metrics: dict*) → dict
Combine the metrics specified in the model and those specified in the config.

Parameters

- **model_metrics** (*dict*) – Metrics as defined by the model

- **config_metrics** (*dict*) – Metrics defined in the config

Returns *dict* – Combined metrics

static _fake_fit(*model: tensorflow.python.keras.engine.training.Model*) → tensorflow.python.keras.engine.training.Model
Fit of the model on fake date to properly initialize the model.

Parameters **model** (*tf.keras.Model*) – Model to initialize

Returns *tf.keras.Model* – Initalized model.

_format_predictions(*predictions: Union[list, numpy.ndarray]*) → dict
Format the predictions to match them with the output names

Parameters **predictions** (*Union[list, np.ndarray]*) – The predictions from the model

Raises **ValueError** – If the predictions do not match with the expected output names

Returns *dict* – Output predictions matched with the output names

_init_data_generators(*labels_only: bool*) → dict
Initialize data generators for all sample folders.

Parameters **labels_only** (*bool*) – Whether to only load labels

Returns *dict* – initalized data generators

static _load_model(*model_file: str, custom_objects: dict*) → Tuple[tensorflow.python.keras.engine.training.Model, ValueError]
Try to load a model, if it doesnt work parse the error.

Parameters

- **model_file** (*str*) – Location of the model file
- **custom_objects** (*dict*) – Potential custom objects to use during model loading

Returns *Tuple[tf.keras.Model, ValueError]* – The model if successfully loaded, otherwise the error

static combine_predictions(*predictions: numpy.ndarray, are_one_hot: bool, label_combination_type: str*) → numpy.ndarray

evaluate()

evaluate_metrics() → dict
Evaluate all metrics for all samples

Returns *dict* – The evaluated metrics

evaluate_metrics_from_predictions(*predictions: dict, real_labels: dict*) → dict
Evaluate the metrics based on the model predictions

Parameters

- **predictions** (*dict*) – Predictions obtained from the model
- **real_labels** (*dict*) – The true labels of the samples for the different outputs

Returns *dict* – The different evaluated metrics

evaluate_sample_metrics() → dict
Evaluate the metrics based on a full sample instead of based on individual batches

Returns *dict* – The evaluated metrics

get_image_output_labels() → dict
Whether an output label is a simple class, the label is actually an image.

Returns *dict* – Output labels that are image outputs

get_real_labels() → dict

get_real_labels_of_sample_subset(*subset_name: str*) → dict
Get the real labels corresponding of all samples from a subset.

Parameters **subset_name** (*str*) – Name of subset to get labels for

Returns *dict* – Real labels for each dataset and output

get_sample_labels_from_patch_labels()

get_sample_predictions_from_patch_predictions()

get_sample_result_from_patch_results(*patch_results*)

get_to_evaluate_metrics() → dict
Get the metrics functions which should be evaluated.

Returns *dict* – Metric function to be evaluated for the different outputs

image_array_to_sitk(*image_array: numpy.ndarray*, *input_name: str*) → SimpleITK.SimpleITK.Image

init_data_generators() → dict
Initialize the data generators.

Returns *dict* – DataGenerator for each subfolder of samples

classmethod init_from_sys_args(*args_in*)

init_label_generators() → dict
Initialize the data generators which only give labels.

Returns *dict* – DataGenerator for each subfolder of samples

init_model_parameters() → None
Initialize the parameters from the model.

static load_model(*model_file: str*, *custom_module: module = None*) → tensorflow.python.keras.engine.training.Model
Load the model, including potential custom losses.

Parameters

- **model_file** (*str*) – Location of the model file
- **custom_module** (*ModuleType*) – Custom module from which to load losses or metrics

Raises **error** – If the model could not be loaded and the problem is not due to a missing loss or metric function.

Returns *tf.keras.Model* – The loaded model

make_dataframe(*sample_names*, *predictions*, *labels*) → pandas.core.frame.DataFrame

make_metric_dataframe(*metrics: dict*) → pandas.core.frame.DataFrame

static one_hot_labels_to_flat_labels(*labels: numpy.ndarray*) → numpy.ndarray

patches_to_sample_image(*datagenerator: PrognosAIs.IO.DataGenerator.HDF5Generator*, *filenames: list*, *output_name: str*, *predictions: numpy.ndarray*, *labels_are_one_hot: bool*, *label_combination_type: str*) → numpy.ndarray

predict() → dict
Get predictions from the model

Returns *dict* – Predictions for the different outputs of the model for all samples

write_image_predictions_to_files(*sample_names*, *predictions*, *labels_one_hot*) → None

write_metrics_to_file() → None

write_predictions_to_file() → None

2.2.5 PrognosAIs.Model.Losses module

class PrognosAIs.Model.Losses.**CoxLoss**(***kwargs*)
Bases: tensorflow.python.keras.losses.Loss

Cox loss as defined in https://arxiv.org/pdf/1606.00931.pdf.

call(*y_true: tensorflow.python.framework.ops.Tensor*, *y_pred: tensorflow.python.framework.ops.Tensor*) → tensorflow.python.framework.ops.Tensor
Calculate the cox loss.

Parameters

- **y_true** (*tf.Tensor*) – Tensor of shape (batch_size, 2), with the first index containing whether and event occurred for each sample, and the second index containing the time to event, or follow-up time if no event has occurred
- **y_pred** (*tf.Tensor*) – The $\hat{h}_{\sigma\}}$ as predicted by the network

Returns *tf.Tensor* – The cox loss for each sample in the batch

get_config() → dict
Get the configuration of the loss.

Returns *dict* – configuration of the loss

class PrognosAIs.Model.Losses.**DICE_loss**(*name: str = 'dice_loss'*, *weighted: bool = False*, *foreground_only: bool = False*, ***kwargs*)
Bases: tensorflow.python.keras.losses.Loss

Loss class for the Sørensen–Dice coefficient.

call(*y_true: tensorflow.python.framework.ops.Tensor*, *y_pred: tensorflow.python.framework.ops.Tensor*) → tensorflow.python.framework.ops.Tensor
Calculate the DICE loss.

This functions calculates the DICE loss defined as:

$$1 - 2 * \frac{|A \cap B|}{|A| + |B|}$$

When no positive labels are found in both A and B the loss returns 0 by default. The loss works both for one-hot predicted labels and binary labels.

Parameters

- **y_true** (*tf.Tensor*) – The ground truth labels, shape: (batch_size, N_1, N_2 ... N_d) where N_d is the number of channels (can be 1). For a 3D tensor with 1 channel (binary class) and batch size of 1 it will have a shape of (1, N_1, N_2, N_3, 1)

- **y_pred** (*tf.Tensor*) – The predicted labels. shape: (batch_size, N_1, N_2 ... N_d) where N_d is the number of channels. When a binary prediction is done (last activation function is sigmoid), N_d = 1. When one-hot prediction are done (last activation function is softmax) N_d = number of classes

Returns *tf.Tensor* – Tensor of length batch_size with the DICE loss for each sample

get_config() → dict
Get the configuration of the loss.

Returns *dict* – configuration of the loss

class PrognosAIs.Model.Losses.**MaskedCategoricalCrossentropy**(*name: str = 'masked_categorical_crossentropy'*, *class_weight: dict = None*, *mask_value: int = - 1*, ***kwargs*)
Bases: tensorflow.python.keras.losses.CategoricalCrossentropy

Caterogical crossentropy loss which takes into account missing values.

__call__(*y_true: tensorflow.python.framework.ops.Tensor*, *y_pred: tensorflow.python.framework.ops.Tensor*, *sample_weight: tensorflow.python.framework.ops.Tensor = None*) → tensorflow.python.framework.ops.Tensor
Obtain the total masked categorical crossentropy loss for the batch.

Parameters

- **y_true** (*tf.Tensor*) – Ground-truth labels, one-hot encoded (batch_size, N_1, N_2, ... N_d) tensor, with N_d the number of outputs
- **y_pred** (*tf.Tensor*) – Predictions one-hot encoded, for example from softmax, (batch_size, N_1, N_2, ... N_d) tensor, with N_d the number of outputs
- **sample_weight** (*tf.Tensor*) – Sample weight for each indidvidual label to be used in reduction of sample loss to overal batch loss

Returns *tf.Tensor* – The total masked categorial crossentropy loss, scalar tensor with rank 0

__init__(*name: str = 'masked_categorical_crossentropy'*, *class_weight: dict = None*, *mask_value: int = - 1*, ***kwargs*) → None
Caterogical crossentropy loss which takes into account missing values.

For the samples with masked values a cross entropy of 0 will be used, for the other samples the standard cross entropy loss will be calculated

Parameters

- **name** (*str*) – Optional name for the op
- **class_weight** (*dict*) – Weights for each class
- **mask_value** (*int*) – The value that indicates that a sample is missing
- ****kwargs** – arguments to pass the default CategoricalCrossentropy loss

call(*y_true: tensorflow.python.framework.ops.Tensor*, *y_pred: tensorflow.python.framework.ops.Tensor*) → tensorflow.python.framework.ops.Tensor
Obtain the masked categorical crossentropy loss for each sample.

Parameters

- **y_true** (*tf.Tensor*) – Ground-truth labels, one-hot encoded (batch_size, N_1, N_2, ... N_d) tensor, with N_d the number of outputs

- **y_pred** (*tf.Tensor*) – Predictions one-hot encoded, for example from softmax, (batch_size, N_1, N_2, ... N_d) tensor, with N_d the number of outputs

Returns

tf.Tensor –

The masked categorial crossentropy loss for each sample, has rank one less than the inputs tensors

get_config() → dict
Get the configuration of the loss.

Returns *dict* – Configuration parameters of the loss

is_unmasked_sample(*y_true: tensorflow.python.framework.ops.Tensor*) → tensorflow.python.framework.ops.Tensor
Get whether the samples are unmasked (i.e. have real label data).

Parameters **y_true** (*tf.Tensor*) – Tensor of the true labels

Returns *tf.Tensor* – Tensor of 0s and 1s indicating whether that sample is unmasked.

2.2.6 PrognosAIs.Model.Metrics module

class PrognosAIs.Model.Metrics.**ConcordanceIndex**(*name='ConcordanceIndex', **kwargs*)
Bases: tensorflow.python.keras.metrics.Metric

result()
Computes and returns the metric value tensor.

Result computation is an idempotent operation that simply calculates the metric value using the state variables.

update_state(*y_true*, *y_pred*, *sample_weight=None*)
Accumulates statistics for the metric.

Note: This function is executed as a graph function in graph mode. This means:

a) Operations on the same resource are executed in textual order. This should make it easier to do things like add the updated value of a variable to another, for example.

b) You don't need to worry about collecting the update ops to execute. All update ops added to the graph by this function will be executed.

As a result, code should generally work the same way with graph or eager execution.

Parameters

- ***args**
- ****kwargs** – A mini-batch of inputs to the Metric.

class PrognosAIs.Model.Metrics.**DICE**(*name='dice_coefficient', foreground_only=True, **kwargs*)
Bases: tensorflow.python.keras.metrics.Metric

get_config()
Returns the serializable config of the metric.

result()
Computes and returns the metric value tensor.

Result computation is an idempotent operation that simply calculates the metric value using the state variables.

update_state(*y_true*, *y_pred*, *sample_weight=None*)
Accumulates statistics for the metric.

Note: This function is executed as a graph function in graph mode. This means:

a) Operations on the same resource are executed in textual order. This should make it easier to do things like add the updated value of a variable to another, for example.

b) You don't need to worry about collecting the update ops to execute. All update ops added to the graph by this function will be executed.

As a result, code should generally work the same way with graph or eager execution.

Parameters

- ***args**
- ****kwargs** – A mini-batch of inputs to the Metric.

class PrognosAIs.Model.Metrics.**MaskedAUC**(*name='MaskedAUC'*, *mask_value=-1*, ***kwargs*)
Bases: tensorflow.python.keras.metrics.AUC

get_config()
Returns the serializable config of the metric.

update_state(*y_true*, *y_pred*, *sample_weight=None*)
Accumulates confusion matrix statistics.

Parameters

- **y_true** – The ground truth values.
- **y_pred** – The predicted values.
- **sample_weight** – Optional weighting of each example. Defaults to 1. Can be a *Tensor* whose rank is either 0, or the same rank as *y_true*, and must be broadcastable to *y_true*.

Returns Update op.

class PrognosAIs.Model.Metrics.**MaskedCategoricalAccuracy**(*name='MaskedCategoricalAccuracy'*, *mask_value=-1*, ***kwargs*)
Bases: tensorflow.python.keras.metrics.CategoricalAccuracy

get_config()
Returns the serializable config of the metric.

update_state(*y_true*, *y_pred*, *sample_weight=None*)
Accumulates metric statistics.

y_true and *y_pred* should have the same shape.

Parameters

- **y_true** – Ground truth values. shape = *[batch_size, d0, .. dN]*.
- **y_pred** – The predicted values. shape = *[batch_size, d0, .. dN]*.

- **sample_weight** – Optional *sample_weight* acts as a coefficient for the metric. If a scalar is provided, then the metric is simply scaled by the given value. If *sample_weight* is a tensor of size *[batch_size]*, then the metric for each sample of the batch is rescaled by the corresponding element in the *sample_weight* vector. If the shape of *sample_weight* is *[batch_size, d0, .. dN-1]* (or can be broadcasted to this shape), then each metric element of *y_pred* is scaled by the corresponding value of *sample_weight*. (Note on *dN-1*: all metric functions reduce by 1 dimension, usually the last axis (-1)).

Returns Update op.

class PrognosAIs.Model.Metrics.**MaskedSensitivity**(*name='masked_sensitivity'*, *mask_value=- 1*, ***kwargs*)

Bases: tensorflow.python.keras.metrics.Metric

reset_states()

Resets all of the metric state variables.

This function is called between epochs/steps, when a metric is evaluated during training.

result()

Computes and returns the metric value tensor.

Result computation is an idempotent operation that simply calculates the metric value using the state variables.

update_state(*y_true*, *y_pred*, *sample_weight=None*)

Accumulates statistics for the metric.

Note: This function is executed as a graph function in graph mode. This means:

a) Operations on the same resource are executed in textual order. This should make it easier to do things like add the updated value of a variable to another, for example.

b) You don't need to worry about collecting the update ops to execute. All update ops added to the graph by this function will be executed.

As a result, code should generally work the same way with graph or eager execution.

Parameters

- ***args**
- ****kwargs** – A mini-batch of inputs to the Metric.

class PrognosAIs.Model.Metrics.**MaskedSpecificity**(*name='masked_specificity'*, *mask_value=- 1*, ***kwargs*)

Bases: tensorflow.python.keras.metrics.Metric

reset_states()

Resets all of the metric state variables.

This function is called between epochs/steps, when a metric is evaluated during training.

result()

Computes and returns the metric value tensor.

Result computation is an idempotent operation that simply calculates the metric value using the state variables.

update_state(*y_true*, *y_pred*, *sample_weight=None*)

Accumulates statistics for the metric.

Note: This function is executed as a graph function in graph mode. This means:

a) Operations on the same resource are executed in textual order. This should make it easier to do things like add the updated value of a variable to another, for example.

b) You don't need to worry about collecting the update ops to execute. All update ops added to the graph by this function will be executed.

As a result, code should generally work the same way with graph or eager execution.

Parameters

- ***args**
- ****kwargs** – A mini-batch of inputs to the Metric.

class PrognosAIs.Model.Metrics.**Sensitivity**(*name='Sensitivity_custom'*, ***kwargs*)
Bases: tensorflow.python.keras.metrics.Metric

reset_states()
Resets all of the metric state variables.

This function is called between epochs/steps, when a metric is evaluated during training.

result()
Computes and returns the metric value tensor.

Result computation is an idempotent operation that simply calculates the metric value using the state variables.

update_state(*y_true*, *y_pred*, *sample_weight=None*)
Accumulates statistics for the metric.

Note: This function is executed as a graph function in graph mode. This means:

a) Operations on the same resource are executed in textual order. This should make it easier to do things like add the updated value of a variable to another, for example.

b) You don't need to worry about collecting the update ops to execute. All update ops added to the graph by this function will be executed.

As a result, code should generally work the same way with graph or eager execution.

Parameters

- ***args**
- ****kwargs** – A mini-batch of inputs to the Metric.

class PrognosAIs.Model.Metrics.**Specificity**(*name='Specificity_custom'*, ***kwargs*)
Bases: tensorflow.python.keras.metrics.Metric

reset_states()
Resets all of the metric state variables.

This function is called between epochs/steps, when a metric is evaluated during training.

result()
Computes and returns the metric value tensor.

Result computation is an idempotent operation that simply calculates the metric value using the state variables.

update_state(*y_true*, *y_pred*, *sample_weight=None*)
Accumulates statistics for the metric.

Note: This function is executed as a graph function in graph mode. This means:

a) Operations on the same resource are executed in textual order. This should make it easier to do things like add the updated value of a variable to another, for example.

b) You don't need to worry about collecting the update ops to execute. All update ops added to the graph by this function will be executed.

As a result, code should generally work the same way with graph or eager execution.

Parameters

- ***args**
- ****kwargs** – A mini-batch of inputs to the Metric.

PrognosAIs.Model.Metrics.**concordance_index**(*y_true*, *y_pred*)

This function determines the concordance index given two tensorflow tensors

y_true contains a label to indicate whether events occurred, and time to events (or time to right censored data if no event occurred)

y_pred is beta*x in the cox model

2.2.7 PrognosAIs.Model.Parsers module

class PrognosAIs.Model.Parsers.**CallbackParser**(*callback_settings: dict*, *root_path: str = None*, *module_paths=None*, *save_name=None*)

Bases: *PrognosAIs.Model.Parsers.StandardParser*

__init__(*callback_settings: dict*, *root_path: str = None*, *module_paths=None*, *save_name=None*)

Parse callback settings to actual callbacks

Parameters **callback_settings** – Settings for the callbacks

Returns None

get_callbacks()

replace_root_path(*settings*, *root_path*)

class PrognosAIs.Model.Parsers.**LossParser**(*loss_settings: dict*, *class_weights: dict = None*, *module_paths=None*)

Bases: *PrognosAIs.Model.Parsers.StandardParser*

__init__(*loss_settings: dict*, *class_weights: dict = None*, *module_paths=None*)

Parse loss settings to actual losses

Parameters **loss_settings** – Settings for the losses

Returns None

get_losses()

class PrognosAIs.Model.Parsers.**MetricParser**(*metric_settings: dict*, *label_names: list = None*, *module_paths=None*)

Bases: *PrognosAIs.Model.Parsers.StandardParser*

__init__(*metric_settings: dict*, *label_names: list = None*, *module_paths=None*) → None

Parse metrics settings to actual metrics

Parameters **loss_settings** – Settings for the losses

convert_metrics_list_to_dict(*metrics: list*) → dict

get_metrics()

class PrognosAIs.Model.Parsers.**OptimizerParser**(*optimizer_settings: dict*, *module_paths=None*)

Bases: *PrognosAIs.Model.Parsers.StandardParser*

__init__(*optimizer_settings: dict*, *module_paths=None*) → None

Interfacing class to easily get a tf.keras.optimizers optimizer

Parameters **optimizer_settings** – Arguments to be passed to the optimizer

Returns None

get_optimizer()

class PrognosAIs.Model.Parsers.**StandardParser**(*config: dict*, *module_paths: list*)

Bases: object

get_class(*class_name*)

parse_settings()

2.2.8 PrognosAIs.Model.Trainer module

class PrognosAIs.Model.Trainer.**Trainer**(*config: PrognosAIs.IO.ConfigLoader.ConfigLoader*, *sample_folder: str*, *output_folder: str*, *tmp_data_folder: Optional[str] = None*, *save_name: Optional[str] = None*)

Bases: object

Trainer to be used for training a model.

__init__(*config: PrognosAIs.IO.ConfigLoader.ConfigLoader*, *sample_folder: str*, *output_folder: str*, *tmp_data_folder: Optional[str] = None*, *save_name: Optional[str] = None*) → None

Trainer to be used for training a model.

Parameters

- **config** (*ConfigLoader.ConfigLoader*) – Config to be used
- **sample_folder** (*str*) – Folder containing the train and validation samples
- **output_folder** (*str*) – Folder to put the resulting model
- **tmp_data_folder** (*str*) – Folder to copy samples to and load from. Defaults to None.
- **save_name** (*str*) – Specify a name to save the model as instead of using a automatically generated one. Defaults to None.

static _get_architecture_name(*model_name: str*, *input_dimensionality: dict*) → Tuple[str, str]

Get the full architecture name from the model name and input dimensionality.

Parameters

- **model_name** (*str*) – Name of the model
- **input_dimensionality** (*dict*) – Dimensionality of the different inputs

Returns *Tuple[str, str]* – Class name of architecture and full achitecture name

_setup_model() → tensorflow.python.keras.engine.training.Model

Get the model architecture from the architecture name (not yet compiled).

Raises **ValueError** – If architecture is not known

Returns *tf.keras.Model* – The loaded architecture

get_distribution_strategy() → tensorflow.python.distribute.distribute_lib.Strategy
Get the appropiate distribution strategy.

A strategy will be returned that can either distribute the training over multiple SLURM nodes, over multi GPUs, train on a single GPU or on a single CPU (in that order).

Returns *tf.distribute.Strategy* – The distribution strategy to be used in training.

classmethod init_from_sys_args(*args_in: list*) → *PrognosAIs.Model.Trainer.Trainer*
Initialize a Trainer object from the command line.

Parameters **args_in** (*list*) – Arguments to parse to the trainer

Returns *Trainer* – The trainer object

load_class_weights() → Union[None, dict]
Load the class weight from the class weight file.

Returns

Union[None, dict] –

Class weights if requested and the class weight file exists, otherwise None.

property model
Model to be used in training.

Returns *tf.keras.Model* – The model

move_data_to_temporary_folder(*data_folder: str*) → str
Move the data to a temporary directory before loading.

Parameters **data_folder** (*str*) – The original data folder

Returns *str* – Folder to which the data has been moved

set_precision_strategy(*float_policy_setting: Union[str, bool]*) → None
Set the appropiate precision strategy for GPUs.

If the GPUs support it a mixed float16 precision will be used (see tf.keras.mixe_precision for more information), which reduces the memory overhead of the training, while doing computation in float32. If GPUs dont support mixed precision, we will try a float16 precision setting. If that doesn't work either the normal policy is used. If you get NaN values for loss or loss doesn't converge it might be because of the policy. Try running the model without a policy setting.

Parameters **float_policy_setting (float_policy_setting** – Union[str, bool]): Which policy to select if set to PrognosAIs.Constants.AUTO, we will automatically determine what can be done. "mixed" will only consider mixed precision, "float16" only considers float16 policy. Set to False to not use a policy

static set_tf_config(*cluster_resolver: tensorflow.python.distribute.cluster_resolver.cluster_resolver.ClusterResolver*, *environment: Optional[str] = None*) → None
Set the TF_CONFIG env variable from the given cluster resolver.

From https://github.com/tensorflow/tensorflow/issues/37693

Parameters

- **cluster_resolver** (*tf.distribute.cluster_resolver.ClusterResolver*) – cluster resolver to use.
- **environment** (*str*) – Environment to set in TF_CONFIG. Defaults to None.

setup_callbacks() → list
Set up callbacks to be used during training.

Returns *list* – the callbacks

setup_data_generator(*sample_folder: str*) → *PrognosAIs.IO.DataGenerator.HDF5Generator*
Set up a data generator for a folder containg train samples.

Parameters **sample_folder** (*str*) – The path to the folder containing the sample files.

Raises **ValueError** – If the sample folder does not exist or does not contain any samples.

Returns *DataGenerator.HDF5Generator* – Datagenerator of the sample in the sample folder.

setup_model() → tensorflow.python.keras.engine.training.Model
Set up model to be used during train.

Returns *tf.keras.Model* – The compiled model to be trained.

property train_data_generator
The train data generator to be used in training.

Returns *DataGenerator.HDF5Generator* – The train data generator

train_model() → str
Train the model.

Returns *str* – The location where the model has been saved

property validation_data_generator
The validation data generator to be used in training.

Returns *DataGenerator.HDF5Generator* – The validation data generator

2.2.9 Module contents

2.3 PrognosAIs.Preprocessing package

2.3.1 Submodules

2.3.2 PrognosAIs.Preprocessing.Preprocessors module

class PrognosAIs.Preprocessing.Preprocessors.**BatchPreprocessor**(*samples_path: str*, *output_directory: str*, *config: dict*)

Bases: object

classmethod init_from_sys_args(*args_in*)

split_into_subsets(*samples: list*, *sample_labels: list*) → dict

start()

class PrognosAIs.Preprocessing.Preprocessors.**SingleSamplePreprocessor**(*sample:* Prognosais.Preprocessing.Samples.In *config: dict, output_directory: str = None*)

Bases: object

static _get_all_images_from_sequence(*image: SimpleITK.SimpleITK.Image, max_dims: int*) → list

Get all of the images from a sequence of images.

Parameters

- **image** (*sitk.Image*) – Multi-dimensional image containg the sequence.
- **max_dims** (*int*) – The number of dimension of each individual image. This should be equal to the dimensionality of the input image - 1. Otherwise, we do not know how to extract the appropiate images

Raises **ValueError** – If the maximum number of dimensions does not fit with the sequences.

Returns *list* – All images extracted from the sequence.

static _get_first_image_from_sequence(*image: SimpleITK.SimpleITK.Image, max_dims: int*) → SimpleITK.SimpleITK.Image

Extract the first image from a sequence of images

Parameters

- **image** (*sitk.Image*) – Multi-dimensional image containg the sequence.
- **max_dims** (*int*) – The maximum number of dimension the output can be.

Returns *sitk.Image* – The first image extracted from the sequence

apply_pipeline(*pipeline=None*)

bias_field_correcting()

build_pipeline() → list

channel_imputation(*sample_channels*)

channels_to_float16(*sample_channels*)

crop_to_mask(*ROI_mask: SimpleITK.SimpleITK.Image, process_masks: bool = True, apply_to_output: bool = False*)

mask_background(*ROI_mask: SimpleITK.SimpleITK.Image, background_value: float = 0.0, process_masks: bool = True, apply_to_output: bool = False*)

static mask_background_to_min(*image, mask*)

masking()

multi_dimension_extracting()

Extract invidiual images from a multi-dimensional sequence.

Raises **NotImplementedError** – If an extraction type is requested that is not supported.

normalizing()

patching() → None

rejecting()

resampling()

saving()

2.3.3 PrognosAIs.Preprocessing.Samples module

class PrognosAIs.Preprocessing.Samples.**ImageSample**(*root_path: str*, *extension_keyword: str = None*, *mask_keyword: str = None*, *labels: dict = None*, *number_of_label_classes: dict = None*, *are_labels_one_hot: bool = False*, *output_channel_names: list = []*, *input_channel_names: list = []*)

Bases: abc.ABC

ImageSample base class

To be implemented by subclasses:

- *init_image_files*: Contains logic for retrieval of channel filepaths
- *load_channels*: Contains logic for loading of channels from filepaths
- *load_output_channels*: Contains logic for loading of output channels from filepaths
- *load_masks*: Contains logic of loading masks from filepaths

Parameters

- **root_path** – Path of the sample. Should contain folders or directories of channels and masks
- **extension_keyword** – Extension of the files to load
- **mask_keyword** (*optional*) – Keyword to identify which filepaths are masks. Defaults to None.

_identify_channel_file(*image_file: str*) → bool
Identify whether an image file should be included as channel

Parameters **image_file** – Image file to check

Returns *bool* – True if image_file is channel, False otherwise

_identify_mask_file(*image_file: str*) → bool
Identify whether an image file is a mask based on the mask keyword

Parameters **image_file** – Image file to check

Returns *bool* – True if image_file is mask, False otherwise

_identify_output_channel_file(*image_file: str*) → bool
Identify whether an image file is a output channel

Parameters **image_file** – Image file to check

Returns *bool* – True if image_file is output channel, False otherwise

_init_channel_files(*image_files: list*) → list
Get only the channel files from the image files, filtering out masks.

Parameters **image_files** (*list*) – Paths to the image files

Returns *list* – The paths to the channel files

_init_mask_files(*image_files: list*) → list
Get only the mask files from the image files, filtering out channels.

Parameters **image_files** (*list*) – Paths to the image files

Returns *list* – The paths to the mask files

_init_output_channel_files(*image_files: list*) → list
Get the output channel files from the image files.

Parameters **image_files** (*list*) – Paths to the image files

Returns *list* – The paths to the output channel files

_parse_function_parameters(*function_parameters*)
Parse the function parameters

Parameters **function_parameters** (*function or tuple*) – Function and possible args and kw_args.

Returns *tuple* – function, args, and kw_args

_perform_sanity_checks()
Automatic sanity check to see if we can process the sample

Raises **NotImplementedError** – If the configuration has not been implemented

add_to_labels(*to_add_labels: List[dict]*, *to_add_number_of_label_classes: dict*) → None

assert_all_channels_same_size()
Check wheter all channels have the same size

Raises **ValueError** – Raised when not all channels have same size

assert_all_masks_same_size()
Check wheter all masks have the same size

Raises **ValueError** – Raised when not all masks have same size

property channel_names
Names of the channels

Returns *list* – Channel names

property channel_size
The image size of the channels

property channels
The channels present in the sample

Channels of a sample can be set by providing either a function, or a tuple consisting of a function, possible function argument and function keyword arguments.

This function will then be applied to all channels in the sample. The function has to output either a SimpleITK Image or a list. In the last case it is assumed that these are patches and the class is updated accordingly

Returns *list* – Channels present in the sample

copy()
Returns a (deep) copy of the instance

Returns *ImageSample* – Deep copy of the instance

static get_appropiate_dtype_from_image(*image: SimpleITK.SimpleITK.Image*) → int
Find the minimum SimpleITK type need to represent the value

Parameters **value** (*float*) – The value to check

Returns *int* – The appropiate SimpleITK to which the value can be casted

static get_appropiate_dtype_from_scalar(*value: Union[int, float]*, *return_np_type: bool = False*) → Union[int, numpy.dtype]
Find the minimum SimpleITK type need to represent the value

Parameters

- **value** (*float*) – The value to check
- **return_np_type** (*bool*) – If True returns the numpy type instead of the SimpleITK type. Defaults to False.

Returns *int* – The appropiate SimpleITK to which the value can be casted

get_example_channel() → SimpleITK.SimpleITK.Image
Provides an example channel of the samples

Returns *sitk.Image* – Single channel of the sample

get_example_channel_patches() → list
Provides an example of all patches of a channel, even if there is only one patch

Returns *list* – Patch(es) of a single channel of the sample

get_example_mask() → SimpleITK.SimpleITK.Image
Provides an example mask of the samples

Returns *sitk.Image* – Single mask of the sample

get_example_mask_patches() → list
Provides an example of all patches of a mask, even if there is only one patch

Returns *list* – Patch(es) of a single mask of the sample

get_example_output_channel() → SimpleITK.SimpleITK.Image
Provides an example output channel of the samples

Returns *sitk.Image* – Single channel of the sample

get_example_output_channel_patches() → list
Provides an example of all patches of a output channel, even if there is only one patch

Returns *list* – Patch(es) of a single output channel of the sample

get_grouped_channels() → list
Groups the channels on a per-patch basis instead of a per-channel basis

The channels property indexes first by channel and then by (possibly) patches. This function instead first indexes by patches (or the whole sample of no patches). This can be handy when all channels are needed at the same time

Returns *list* – Grouped channels for each patch

get_grouped_masks() → list
Groups the masks on a per-patch basis instead of a per-channel basis

The masks property indexes first by channel and then by (possibly) patches. This function instead first indexes by patches (or the whole sample of no patches). This can be handy when all masks are needed at the same time

Returns *list* – Grouped channels for each patch. Empty lists if sample doesn't have mask

get_grouped_output_channels() → list
Groups the output channels on a per-patch basis instead of a per-channel basis

The channels property indexes first by channel and then by (possibly) patches. This function instead first indexes by patches (or the whole sample of no patches). This can be handy when all channels are needed at the same time

Returns *list* – Grouped channels for each patch

static get_numpy_type_from_sitk_type(*sitk_type: int*) → numpy.dtype

static get_sitk_type_from_numpy_type(*numpy_type: numpy.dtype*) → int

abstract init_image_files() → list
Get the filepaths (folders or files) of the channels for a single sample. To be implemented by the subclass

Returns *list* – The filepaths of the channels

abstract load_channels(*channel_files: list*) → dict
Load the channels from the channel files. To be implemented by the subclass

Example subclass implementation:

```
def load_channels(self, channel_files):
    channels = {}

    nifti_reader = sitk.ImageFileReader()
    nifti_reader.SetImageIO("NiftiImageIO")
    for i_channel_file in channel_files:
        nifti_reader.SetFileName(i_channel_file)
        i_channel = nifti_reader.Execute()
        i_channel_name = os.path.basename(i_channel_file)
        channels[i_channel_name] = i_channel
    return channels
```

Parameters **channel_files** (*list*) – Paths to the channels to be loaded

Returns *dict* – mapping the channel file to the loaded image

abstract load_masks(*mask_files: list*) → dict
Load the masks from the mask files. To be implemented by the subclass

Example subclass implementation:

```
def load_masks(self, mask_files):
    masks = {}
    nifti_reader = sitk.ImageFileReader()
    nifti_reader.SetImageIO("NiftiImageIO")
    for i_mask_file in mask_files:
        nifti_reader.SetFileName(i_mask_file)
        i_mask = nifti_reader.Execute()
        i_mask = sitk.Cast(i_mask, sitk.sitkUInt8)
        i_mask_name = IO_utils.get_file_name(i_mask_file, self.image_
↪extension)
```

(continues on next page)

(continued from previous page)

```
        masks[i_mask_name] = i_mask
    return masks
```

Parameters **mask_files** (*list*) – Paths to the masks to be loaded

Returns *dict* – mapping the mask file to the loaded mask

property mask_names
Names of the masks

Returns *list* – Mask names

property mask_size
The image size of the masks.

property masks
The masks present in the sample

Masks of a sample can be set by providing either a function, or a tuple consisting of a function, possible function argument and function keyword arguments.

This function will then be applied to all masks in the sample. The function has to output either a SimpleITK Image or a list. In the last case it is assumed that these are patches and the class is updated accordingly

Returns *list* – masks present in the sample

property output_channel_size
The image size of the channels

property output_channels
The output channels present in the sample

Output channels of a sample can be set by providing either a function, or a tuple consisting of a function, possible function argument and function keyword arguments.

This function will then be applied to all output channels in the sample. The function has to output either a SimpleITK Image or a list. In the last case it is assumed that these are patches and the class is updated accordingly

Returns *list* – Channels present in the sample

static promote_simpleitk_types(*type_1: int*, *type_2: int*) → int
Get the datatype that can represent both datatypes

Parameters

- **type_1** (*int*) – SimpleITK datype of variable 1
- **type_2** (*int*) – SimpleITK datatype of variable 2

Returns *int* – SimpleITK datatype that can represent both datatypes

update_channel_size()
Update the channel size according to the current channels

update_labels() → None

update_mask_size() → None
Update the mask size according to the current masks

update_metadata() → None

update_output_channel_size()
Update the channel size according to the current channels

class PrognosAIs.Preprocessing.Samples.**NIFTISample**(***kwds*)
Bases: *PrognosAIs.Preprocessing.Samples.ImageSample*

init_image_files()
Get the filepaths (folders or files) of the channels for a single sample. To be implemented by the subclass

Returns *list* – The filepaths of the channels

load_channels(*channel_files*)
Load the channels from the channel files. To be implemented by the subclass

Example subclass implementation:

```
def load_channels(self, channel_files):
    channels = {}

    nifti_reader = sitk.ImageFileReader()
    nifti_reader.SetImageIO("NiftiImageIO")
    for i_channel_file in channel_files:
        nifti_reader.SetFileName(i_channel_file)
        i_channel = nifti_reader.Execute()
        i_channel_name = os.path.basename(i_channel_file)
        channels[i_channel_name] = i_channel
    return channels
```

Parameters **channel_files** (*list*) – Paths to the channels to be loaded

Returns *dict* – mapping the channel file to the loaded image

load_masks(*mask_files*)
Load the masks from the mask files. To be implemented by the subclass

Example subclass implementation:

```
def load_masks(self, mask_files):
    masks = {}
    nifti_reader = sitk.ImageFileReader()
    nifti_reader.SetImageIO("NiftiImageIO")
    for i_mask_file in mask_files:
        nifti_reader.SetFileName(i_mask_file)
        i_mask = nifti_reader.Execute()
        i_mask = sitk.Cast(i_mask, sitk.sitkUInt8)
        i_mask_name = IO_utils.get_file_name(i_mask_file, self.image_
↪extension)
        masks[i_mask_name] = i_mask
    return masks
```

Parameters **mask_files** (*list*) – Paths to the masks to be loaded

Returns *dict* – mapping the mask file to the loaded mask

PrognosAIs.Preprocessing.Samples.**get_sample_class**(*sample_type_name: str*)

2.3.4 Module contents

CHAPTER

THREE

SUBMODULES

CHAPTER

FOUR

PROGNOSAIS.CONSTANTS MODULE

CHAPTER

FIVE

PROGNOSAIS.PIPELINE MODULE

class PrognosAIs.Pipeline.**Pipeline**(*config_file: str*, *preprocess: bool = True*, *train: bool = True*, *evaluate: bool = True*, *samples_folder: str = None*)

Bases: object

classmethod init_from_sys_args(*args_in*)

start_local_pipeline()

start_slurm_pipeline(*preprocess_job: slurmpie.slurmpie.Job*, *train_job: slurmpie.slurmpie.Job*, *evaluate_job: slurmpie.slurmpie.Job*)

CHAPTER

SIX

MODULE CONTENTS

PYTHON MODULE INDEX

p

PrognosAIs, 57
PrognosAIs.Constants, 53
PrognosAIs.IO, 19
PrognosAIs.IO.ConfigLoader, 7
PrognosAIs.IO.Configs, 9
PrognosAIs.IO.DataGenerator, 10
PrognosAIs.IO.LabelParser, 16
PrognosAIs.IO.utils, 18
PrognosAIs.Model, 41
PrognosAIs.Model.Architectures, 29
PrognosAIs.Model.Architectures.AlexNet, 19
PrognosAIs.Model.Architectures.Architecture, 20
PrognosAIs.Model.Architectures.DDSNet, 21
PrognosAIs.Model.Architectures.DenseNet, 22
PrognosAIs.Model.Architectures.InceptionNet, 25
PrognosAIs.Model.Architectures.ResNet, 26
PrognosAIs.Model.Architectures.UNet, 27
PrognosAIs.Model.Architectures.VGG, 28
PrognosAIs.Model.Callbacks, 29
PrognosAIs.Model.Evaluators, 29
PrognosAIs.Model.Losses, 32
PrognosAIs.Model.Metrics, 34
PrognosAIs.Model.Parsers, 38
PrognosAIs.Model.Trainer, 39
PrognosAIs.Pipeline, 55
PrognosAIs.Preprocessing, 49
PrognosAIs.Preprocessing.Preprocessors, 41
PrognosAIs.Preprocessing.Samples, 43

INDEX

Symbols

__call__() (*PrognosAIs.Model.Losses.MaskedCategoricalCrossentropy method*), 33
__init__() (*PrognosAIs.IO.DataGenerator.Augmentor method*), 10
__init__() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 12
__init__() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 16
__init__() (*PrognosAIs.Model.Losses.MaskedCategoricalCrossentropy method*), 33
__init__() (*PrognosAIs.Model.Parsers.CallbackParser method*), 38
__init__() (*PrognosAIs.Model.Parsers.LossParser method*), 38
__init__() (*PrognosAIs.Model.Parsers.MetricParser method*), 38
__init__() (*PrognosAIs.Model.Parsers.OptimizerParser method*), 39
__init__() (*PrognosAIs.Model.Trainer.Trainer method*), 39
_combine_config_and_model_metrics() (*PrognosAIs.Model.Evaluators.Evaluator method*), 29
_fake_fit() (*PrognosAIs.Model.Evaluators.Evaluator static method*), 30
_format_predictions() (*PrognosAIs.Model.Evaluators.Evaluator method*), 30
_get_all_dataset_attributes() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 13
_get_all_images_from_sequence() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor static method*), 42
_get_architecture_name() (*PrognosAIs.Model.Trainer.Trainer static method*), 39
_get_dataset_names() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 13
_get_first_image_from_sequence() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor static method*), 42
_identify_channel_file() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 43
_identify_mask_file() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 43
_identify_output_channel_file() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 43
_init_channel_files() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 43
_init_data_generators() (*PrognosAIs.Model.Evaluators.Evaluator method*), 30
_init_mask_files() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
_init_output_channel_files() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
_load_model() (*PrognosAIs.Model.Evaluators.Evaluator static method*), 30
_parse_function_parameters() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
_perform_sanity_checks() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
_setup_model() (*PrognosAIs.Model.Trainer.Trainer method*), 39

A

add_to_labels() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
AlexNet_2D (*class in PrognosAIs.Model.Architectures.AlexNet*), 19
AlexNet_3D (*class in PrognosAIs.Model.Architectures.AlexNet*), 20
apply_augmentation() (*PrognosAIs.IO.DataGenerator.Augmentor method*), 11
apply_augmentation() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 13
apply_pipeline() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
assert_all_channels_same_size() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
assert_all_masks_same_size() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
augment_sample() (*PrognosAIs.IO.DataGenerator.Augmentor method*), 11
Augmentor (*class in PrognosAIs.IO.DataGenerator*), 10

B

BatchPreprocessor (*class in PrognosAIs.Preprocessing.Preprocessors*), 41
bias_field_correcting() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
bias_field_correcting_config (*class in PrognosAIs.IO.Configs*), 9
build_pipeline() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42

C

calculate_concordance_index() (*in module PrognosAIs.Model.Callbacks*), 29
call() (*PrognosAIs.Model.Losses.CoxLoss method*), 32
call() (*PrognosAIs.Model.Losses.DICE_loss method*), 32
call() (*PrognosAIs.Model.Losses.MaskedCategoricalCrossentropy method*), 33
CallbackParser (*class in PrognosAIs.Model.Parsers*), 38
channel_imputation() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
channel_names() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 44
channel_size() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 44
channels() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 44
channels_to_float16() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
check_minimum_input_size() (*PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture static method*), 20
ClassificationNetworkArchitecture (*class in PrognosAIs.Model.Architectures.Architecture*), 20
combine_predictions() (*PrognosAIs.Model.Evaluators.Evaluator static method*), 30
concordance_index() (*in module PrognosAIs.Model.Metrics*), 38
ConcordanceIndex (*class in PrognosAIs.Model.Callbacks*), 29
ConcordanceIndex (*class in PrognosAIs.Model.Metrics*), 34
config (*class in PrognosAIs.IO.Configs*), 9
ConfigLoader (*class in PrognosAIs.IO.ConfigLoader*), 7
convert_metrics_list_to_dict() (*PrognosAIs.Model.Parsers.MetricParser method*), 38
copy() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 44
copy_config() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
copy_directory() (*in module PrognosAIs.IO.utils*), 18
CoxLoss (*class in PrognosAIs.Model.Losses*), 32
create_directory() (*in module PrognosAIs.IO.utils*), 18
create_model() (*PrognosAIs.Model.Architectures.AlexNet.AlexNet_2D method*), 19
create_model() (*PrognosAIs.Model.Architectures.AlexNet.AlexNet_3D method*), 20
create_model() (*PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture method*), 20
create_model() (*Progno-*

sAIs.Model.Architectures.DDSNet.DDSNet_2D method), 21
create_model() (*PrognosAIs.Model.Architectures.DDSNet.DDSNet_3D method*), 21
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_2D method*), 22
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_3D method*), 22
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_2D method*), 23
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_3D method*), 23
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_2D method*), 23
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_3D method*), 24
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_2D method*), 24
create_model() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_3D method*), 24
create_model() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionNet_InceptionResNetV2_2D method*), 25
create_model() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionNet_InceptionResNetV2_3D method*), 25
create_model() (*PrognosAIs.Model.Architectures.ResNet.ResNet_18_2D method*), 26
create_model() (*PrognosAIs.Model.Architectures.ResNet.ResNet_18_3D method*), 26
create_model() (*PrognosAIs.Model.Architectures.ResNet.ResNet_18_multioutput_3D method*), 26
create_model() (*PrognosAIs.Model.Architectures.ResNet.ResNet_34_2D method*), 26
create_model() (*PrognosAIs.Model.Architectures.ResNet.ResNet_34_3D method*), 27
create_model() (*PrognosAIs.Model.Architectures.UNet.UNet_2D method*), 27
create_model() (*PrognosAIs.Model.Architectures.UNet.UNet_3D method*), 27
create_model() (*PrognosAIs.Model.Architectures.VGG.VGG_16_2D method*), 28
create_model() (*PrognosAIs.Model.Architectures.VGG.VGG_16_3D method*), 28
create_model() (*PrognosAIs.Model.Architectures.VGG.VGG_19_2D method*), 28
create_model() (*PrognosAIs.Model.Architectures.VGG.VGG_19_3D method*), 28
crop_to_mask() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42

D

DDSNet (*class in PrognosAIs.Model.Architectures.DDSNet*), 21
DDSNet_2D (*class in PrognosAIs.Model.Architectures.DDSNet*), 21
DDSNet_3D (*class in PrognosAIs.Model.Architectures.DDSNet*), 21
delete_directory() (*in module PrognosAIs.IO.utils*), 18
DenseNet (*class in PrognosAIs.Model.Architectures.DenseNet*), 22
DenseNet_121_2D (*class in PrognosAIs.Model.Architectures.DenseNet*), 22
DenseNet_121_3D (*class in PrognosAIs.Model.Architectures.DenseNet*), 22
DenseNet_169_2D (*class in PrognosAIs.Model.Architectures.DenseNet*), 22
DenseNet_169_3D (*class in PrognosAIs.Model.Architectures.DenseNet*), 23
DenseNet_201_2D (*class in PrognosAIs.Model.Architectures.DenseNet*), 23
DenseNet_201_3D (*class in PrognosAIs.Model.Architectures.DenseNet*), 23
DenseNet_264_2D (*class in PrognosAIs.Model.Architectures.DenseNet*), 24
DenseNet_264_3D (*class in PrognosAIs.Model.Architectures.DenseNet*), 24
DICE (*class in PrognosAIs.Model.Metrics*), 34
DICE_loss (*class in PrognosAIs.Model.Losses*), 32
dims (*PrognosAIs.Model.Architectures.DDSNet.DDSNet_2D attribute*), 21
dims (*PrognosAIs.Model.Architectures.DDSNet.DDSNet_3D attribute*), 21
dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_2D attribute*), 22

dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_3D attribute*), 22
dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_2D attribute*), 23
dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_3D attribute*), 23
dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_2D attribute*), 23
dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_3D attribute*), 24
dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_2D attribute*), 24
dims (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_3D attribute*), 24
dims (*PrognosAIs.Model.Architectures.UNet.UNet_2D attribute*), 27
dims (*PrognosAIs.Model.Architectures.UNet.UNet_3D attribute*), 27
dims (*PrognosAIs.Model.Architectures.VGG.VGG_16_2D attribute*), 28
dims (*PrognosAIs.Model.Architectures.VGG.VGG_16_3D attribute*), 28
dims (*PrognosAIs.Model.Architectures.VGG.VGG_19_2D attribute*), 28
dims (*PrognosAIs.Model.Architectures.VGG.VGG_19_3D attribute*), 28

E

encode_labels_one_hot() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
evaluate() (*PrognosAIs.Model.Evaluators.Evaluator method*), 30
evaluate_metrics() (*PrognosAIs.Model.Evaluators.Evaluator method*), 30
evaluate_metrics_from_predictions() (*PrognosAIs.Model.Evaluators.Evaluator method*), 30
evaluate_sample_metrics() (*PrognosAIs.Model.Evaluators.Evaluator method*), 30
Evaluator (*class in PrognosAIs.Model.Evaluators*), 29

F

feature_loader() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 13
features_and_labels_loader() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 13
find_files_with_extension() (*in module PrognosAIs.IO.utils*), 18
fit_in_memory() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14

G

general_config (*class in PrognosAIs.IO.Configs*), 9
get_all_dataset_attributes() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_appropiate_dtype_from_image() (*PrognosAIs.Preprocessing.Samples.ImageSample static method*), 45
get_appropiate_dtype_from_scalar() (*PrognosAIs.Preprocessing.Samples.ImageSample static method*), 45
get_available_ram() (*in module PrognosAIs.IO.utils*), 18
get_batch_size() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_cache_in_memory() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_callback_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_callbacks() (*PrognosAIs.Model.Parsers.CallbackParser method*), 38
get_center_patch_around_mask() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_class() (*PrognosAIs.Model.Parsers.StandardParser method*), 39
get_class_weights() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_class_weights() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
get_cluster_setting() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_cluster_type() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_combine_patch_predictions() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_config() (*PrognosAIs.Model.Losses.CoxLoss method*), 32
get_config() (*PrognosAIs.Model.Losses.DICE_loss method*), 33

get_config() (*PrognosAIs.Model.Losses.MaskedCategoricalCrossentropy method*), 34
get_config() (*PrognosAIs.Model.Metrics.DICE method*), 34
get_config() (*PrognosAIs.Model.Metrics.MaskedAUC method*), 35
get_config() (*PrognosAIs.Model.Metrics.MaskedCategoricalAccuracy method*), 35
get_config_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_conv_block() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
get_copy_files() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_corrected_stride_size() (*PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture static method*), 20
get_cpu_devices() (*in module PrognosAIs.IO.utils*), 18
get_cropping_block() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
get_custom_definitions_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_data() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
get_data_augmentation() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_data_augmentation_factor() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_data_augmentation_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_data_folder() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_dataset_attribute() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_dataset_distribution() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_dataset_names() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_DDS_block() (*PrognosAIs.Model.Architectures.DDSNet.DDSNet method*), 21
get_dense_block() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet method*), 22
get_dense_stem() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet method*), 22
get_depth() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
get_dir_size() (*in module PrognosAIs.IO.utils*), 19
get_distribution_strategy() (*PrognosAIs.Model.Trainer.Trainer method*), 40
get_do_augmentation() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_dtype() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_evaluate_metrics() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_evaluate_train_set() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_evaluation_mask_labels() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_evaluation_metric_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_example_channel() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_example_channel_patches() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_example_mask() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_example_mask_patches() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_example_output_channel() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_example_output_channel_patches() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_extra_input_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8

get_feature_attribute() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_feature_dimensionality() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_feature_metadata() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_feature_metadata_from_sample() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_feature_shape() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_feature_size() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 14
get_file_name() (*in module PrognosAIs.IO.utils*), 19
get_file_name_from_full_path() (*in module PrognosAIs.IO.utils*), 19
get_file_path() (*in module PrognosAIs.IO.utils*), 19
get_filter_missing() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_float16_epsilon() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_float_policy() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_fsl_reorient_bin() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_fsl_val_bin() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_gpu_compute_capability() (*in module PrognosAIs.IO.utils*), 19
get_gpu_devices() (*in module PrognosAIs.IO.utils*), 19
get_gpu_workers() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_grouped_channels() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_grouped_masks() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 45
get_grouped_output_channels() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 46
get_image_output_labels() (*PrognosAIs.Model.Evaluators.Evaluator method*), 30
get_image_size() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_inception_resnet_A() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet method*), 25
get_inception_resnet_B() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet method*), 25
get_inception_resnet_C() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet method*), 25
get_inception_resnet_reduction_A() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet method*), 25
get_inception_resnet_reduction_B() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet method*), 25
get_inception_stem() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet method*), 25
get_input_folder() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_keep_rejected_patches() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_label_attribute() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
get_label_categories() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
get_label_category_type() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
get_label_combination_type() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_label_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_label_from_sample() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
get_labels() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
get_labels_are_one_hot() (*PrognosAIs.IO.DataGenerator.HDF5Generator*

method), 15
get_labels_from_category() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 17
get_loss_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_loss_weights() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_losses() (*PrognosAIs.Model.Parsers.LossParser method*), 38
get_make_one_hot() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_make_patches() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_mask_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_mask_keyword() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_max_steps_per_epoch() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_metric_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_metrics() (*PrognosAIs.Model.Parsers.MetricParser method*), 39
get_min_patch_voxels() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_model_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_model_name() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_model_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_multi_channels_patches() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_N_classes() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_N_epoch() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_N_jobs() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_N_max_patches() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 7
get_number_of_channels() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
get_number_of_classes() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
get_number_of_classes() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 18
get_number_of_classes_from_category() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 18
get_number_of_cpus() (*in module PrognosAIs.IO.utils*), 19
get_number_of_filters() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
get_number_of_gpu_devices() (*in module PrognosAIs.IO.utils*), 19
get_number_of_samples() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 18
get_number_of_slurm_nodes() (*in module PrognosAIs.IO.utils*), 19
get_numpy_iterator() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
get_numpy_type_from_sitk_type() (*PrognosAIs.Preprocessing.Samples.ImageSample static method*), 46
get_optimizer() (*PrognosAIs.Model.Parsers.OptimizerParser method*), 39
get_optimizer_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_original_label_category_type() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 18
get_original_labels_from_category() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 18
get_output_folder() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_padding_block() (*PrognosAIs.Model.Architectures.UNet.Unet method*),

27
get_parent_directory() (*in module PrognosAIs.IO.utils*), 19
get_patch_predictions() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_patch_size() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_pool_block() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
get_preprocessings_settings() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_processed_samples_folder() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 8
get_random_state() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_real_labels() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
get_real_labels_of_sample_subset() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
get_reject_patches() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_resample_images() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_resample_size() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_rescale_mask_intensity() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_residual_conv_block() (*PrognosAIs.Model.Architectures.ResNet.ResNet method*), 26
get_residual_identity_block() (*PrognosAIs.Model.Architectures.ResNet.ResNet method*), 26
get_resume_training_from_model() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_root_name() (*in module PrognosAIs.IO.utils*), 19
get_sample_class() (*in module PrognosAIs.Preprocessing.Samples*), 48
get_sample_labels_from_patch_labels() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
get_sample_predictions_from_patch_predictions() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
get_sample_result_from_patch_results() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
get_samples() (*PrognosAIs.IO.LabelParser.LabelLoader method*), 18
get_save_name() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_seed() (*PrognosAIs.IO.DataGenerator.Augmentor method*), 11
get_shuffle() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_shuffle_evaluation() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_shuffle_val() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_sitk_type_from_numpy_type() (*PrognosAIs.Preprocessing.Samples.ImageSample static method*), 46
get_size_string() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_spec() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
get_specific_output_folder() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_step_type() (*PrognosAIs.IO.Configs.config static method*), 9
get_stratify_index() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_subdirectories() (*in module PrognosAIs.IO.utils*), 19
get_test_data_folder() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_test_label_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_test_model_file() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_tf_dataset() (*Progno-*

sAIs.IO.DataGenerator.HDF5Generator method), 15
get_to_evaluate_metrics() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
get_training_multi_processing() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_transition_block() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet method*), 22
get_upsampling_block() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
get_use_class_weights() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_use_class_weights_in_losses() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_use_labels_from_rejection() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_use_mask_as_channel() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_use_mask_as_label() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
get_VGG_block() (*PrognosAIs.Model.Architectures.VGG.VGG method*), 28
get_write_predictions() (*PrognosAIs.IO.ConfigLoader.ConfigLoader method*), 9
gpu_supports_float16() (*in module PrognosAIs.IO.utils*), 19
gpu_supports_mixed_precision() (*in module PrognosAIs.IO.utils*), 19
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_2D attribute*), 22
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_3D attribute*), 22
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_2D attribute*), 23
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_3D attribute*), 23
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_2D attribute*), 23
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_3D attribute*), 24
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_2D attribute*), 24
GROWTH_RATE (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_3D attribute*), 24

H

HDF5Generator (*class in PrognosAIs.IO.DataGenerator*), 12

I

image_array_to_sitk() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
ImageSample (*class in PrognosAIs.Preprocessing.Samples*), 43
InceptionNet_InceptionResNetV2_2D (*class in PrognosAIs.Model.Architectures.InceptionNet*), 25
InceptionNet_InceptionResNetV2_3D (*class in PrognosAIs.Model.Architectures.InceptionNet*), 25
InceptionResNet (*class in PrognosAIs.Model.Architectures.InceptionNet*), 25
init_data_generators() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
init_dimensionality() (*PrognosAIs.Model.Architectures.DDSNet.DDSNet method*), 21
init_dimensionality() (*PrognosAIs.Model.Architectures.DenseNet.DenseNet method*), 22
init_dimensionality() (*PrognosAIs.Model.Architectures.InceptionNet.InceptionResNet method*), 25
init_dimensionality() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
init_dimensionality() (*PrognosAIs.Model.Architectures.VGG.VGG method*), 28
init_from_sys_args() (*PrognosAIs.Model.Evaluators.Evaluator class method*), 31
init_from_sys_args() (*PrognosAIs.Model.Trainer.Trainer class method*), 40

init_from_sys_args() (*PrognosAIs.Pipeline.Pipeline class method*), 55
init_from_sys_args() (*PrognosAIs.Preprocessing.Preprocessors.BatchPreprocessor class method*), 41
init_image_files() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 46
init_image_files() (*PrognosAIs.Preprocessing.Samples.NIFTISample method*), 48
init_label_generators() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
init_model_parameters() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_2D attribute*), 22
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_3D attribute*), 22
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_2D attribute*), 23
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_3D attribute*), 23
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_2D attribute*), 23
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_3D attribute*), 24
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_2D attribute*), 24
INITIAL_FILTERS (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_3D attribute*), 24
is_unmasked_sample() (*PrognosAIs.Model.Losses.MaskedCategoricalCrossentropy method*), 34

L

label_loader() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
labeling_config (*class in PrognosAIs.IO.Configs*), 9
LabelLoader (*class in PrognosAIs.IO.LabelParser*), 16
load_channels() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 46
load_channels() (*PrognosAIs.Preprocessing.Samples.NIFTISample method*), 48
load_class_weights() (*PrognosAIs.Model.Trainer.Trainer method*), 40
load_features() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
load_labels() (*PrognosAIs.IO.DataGenerator.HDF5Generator method*), 15
load_masks() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 46
load_masks() (*PrognosAIs.Preprocessing.Samples.NIFTISample method*), 48
load_model() (*PrognosAIs.Model.Evaluators.Evaluator static method*), 31
load_module_from_file() (*in module PrognosAIs.IO.utils*), 19
LossParser (*class in PrognosAIs.Model.Parsers*), 38

M

make_dataframe() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
make_dropout_layer() (*PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture method*), 21
make_inputs() (*PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture method*), 21
make_metric_dataframe() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
make_outputs() (*PrognosAIs.Model.Architectures.Architecture.ClassificationNetworkArch static method*), 20
make_outputs() (*PrognosAIs.Model.Architectures.Architecture.NetworkArchitecture method*), 21
make_outputs() (*PrognosAIs.Model.Architectures.UNet.Unet method*), 27
mask() (*PrognosAIs.IO.Configs.bias_field_correcting_config property*), 9
mask() (*PrognosAIs.IO.Configs.masking_config property*), 10
mask() (*PrognosAIs.IO.Configs.normalizing_config property*), 10

mask() (*PrognosAIs.IO.Configs.rejecting_config property*), 10
mask_background() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
mask_background_to_min() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor static method*), 42
mask_file() (*PrognosAIs.IO.Configs.bias_field_correcting_config property*), 9
mask_file() (*PrognosAIs.IO.Configs.masking_config property*), 10
mask_file() (*PrognosAIs.IO.Configs.normalizing_config property*), 10
mask_file() (*PrognosAIs.IO.Configs.rejecting_config property*), 10
mask_names() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 47
mask_size() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 47
MaskedAUC (*class in PrognosAIs.Model.Metrics*), 35
MaskedCategoricalAccuracy (*class in PrognosAIs.Model.Metrics*), 35
MaskedCategoricalCrossentropy (*class in PrognosAIs.Model.Losses*), 33
MaskedSensitivity (*class in PrognosAIs.Model.Metrics*), 36
MaskedSpecificity (*class in PrognosAIs.Model.Metrics*), 36
masking() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
masking_config (*class in PrognosAIs.IO.Configs*), 9
masks() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 47
MetricParser (*class in PrognosAIs.Model.Parsers*), 38
model() (*PrognosAIs.Model.Trainer.Trainer property*), 40
module
    PrognosAIs, 57
    PrognosAIs.Constants, 53
    PrognosAIs.IO, 19
    PrognosAIs.IO.ConfigLoader, 7
    PrognosAIs.IO.Configs, 9
    PrognosAIs.IO.DataGenerator, 10
    PrognosAIs.IO.LabelParser, 16
    PrognosAIs.IO.utils, 18
    PrognosAIs.Model, 41
    PrognosAIs.Model.Architectures, 29
    PrognosAIs.Model.Architectures.AlexNet, 19
    PrognosAIs.Model.Architectures.Architecture, 20
    PrognosAIs.Model.Architectures.DDSNet, 21
    PrognosAIs.Model.Architectures.DenseNet, 22
    PrognosAIs.Model.Architectures.InceptionNet, 25
    PrognosAIs.Model.Architectures.ResNet, 26
    PrognosAIs.Model.Architectures.UNet, 27
    PrognosAIs.Model.Architectures.VGG, 28
    PrognosAIs.Model.Callbacks, 29
    PrognosAIs.Model.Evaluators, 29
    PrognosAIs.Model.Losses, 32
    PrognosAIs.Model.Metrics, 34
    PrognosAIs.Model.Parsers, 38
    PrognosAIs.Model.Trainer, 39
    PrognosAIs.Pipeline, 55
    PrognosAIs.Preprocessing, 49
    PrognosAIs.Preprocessing.Preprocessors, 41
    PrognosAIs.Preprocessing.Samples, 43
move_data_to_temporary_folder() (*PrognosAIs.Model.Trainer.Trainer method*), 40
multi_dimension_extracting() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
multi_dimension_extracting_config (*class in PrognosAIs.IO.Configs*), 10

N

NetworkArchitecture (*class in PrognosAIs.Model.Architectures.Architecture*), 20
NIFTISample (*class in PrognosAIs.Preprocessing.Samples*), 48
normalize_path() (*in module PrognosAIs.IO.utils*), 19
normalizing() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 42
normalizing_config (*class in PrognosAIs.IO.Configs*), 10

O

on_epoch_end() (*PrognosAIs.Model.Callbacks.ConcordanceIndex method*), 29
on_epoch_end() (*PrognosAIs.Model.Callbacks.Timer method*), 29

one_hot_labels_to_flat_labels() (*PrognosAIs.Model.Evaluators.Evaluator static method*), 31
OptimizerParser (*class in PrognosAIs.Model.Parsers*), 39
output_channel_size() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 47
output_channels() (*PrognosAIs.Preprocessing.Samples.ImageSample property*), 47

P

pad_to_original_size() (*PrognosAIs.IO.DataGenerator.Augmentor method*), 11
padding_type (*PrognosAIs.Model.Architectures.AlexNet.AlexNet_2D attribute*), 20
padding_type (*PrognosAIs.Model.Architectures.AlexNet.AlexNet_3D attribute*), 20
parse_settings() (*PrognosAIs.Model.Parsers.StandardParser method*), 39
patch_size() (*PrognosAIs.IO.Configs.patching_config property*), 10
patches_to_sample_image() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
patching() (*PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method*), 43
patching_config (*class in PrognosAIs.IO.Configs*), 10
Pipeline (*class in PrognosAIs.Pipeline*), 55
predict() (*PrognosAIs.Model.Evaluators.Evaluator method*), 31
PrognosAIs
    module, 57
PrognosAIs.Constants
    module, 53
PrognosAIs.IO
    module, 19
PrognosAIs.IO.ConfigLoader
    module, 7
PrognosAIs.IO.Configs
    module, 9
PrognosAIs.IO.DataGenerator
    module, 10
PrognosAIs.IO.LabelParser
    module, 16
PrognosAIs.IO.utils
    module, 18
PrognosAIs.Model
    module, 41
PrognosAIs.Model.Architectures
    module, 29
PrognosAIs.Model.Architectures.AlexNet
    module, 19
PrognosAIs.Model.Architectures.Architecture
    module, 20
PrognosAIs.Model.Architectures.DDSNet
    module, 21
PrognosAIs.Model.Architectures.DenseNet
    module, 22
PrognosAIs.Model.Architectures.InceptionNet
    module, 25
PrognosAIs.Model.Architectures.ResNet
    module, 26
PrognosAIs.Model.Architectures.UNet
    module, 27
PrognosAIs.Model.Architectures.VGG
    module, 28
PrognosAIs.Model.Callbacks
    module, 29
PrognosAIs.Model.Evaluators
    module, 29
PrognosAIs.Model.Losses
    module, 32
PrognosAIs.Model.Metrics
    module, 34
PrognosAIs.Model.Parsers
    module, 38
PrognosAIs.Model.Trainer
    module, 39
PrognosAIs.Pipeline
    module, 55
PrognosAIs.Preprocessing
    module, 49
PrognosAIs.Preprocessing.Preprocessors
    module, 41
PrognosAIs.Preprocessing.Samples
    module, 43
promote_simpleitk_types() (*PrognosAIs.Preprocessing.Samples.ImageSample static method*), 47

R

random_brightness() (*PrognosAIs.IO.DataGenerator.Augmentor method*), 11
random_contrast() (*PrognosAIs.IO.DataGenerator.Augmentor method*), 12
random_cropping() (*PrognosAIs.IO.DataGenerator.Augmentor method*),

12
random_flipping() (PrognosAIs.IO.DataGenerator.Augmentor method), 12
random_rotate() (PrognosAIs.IO.DataGenerator.Augmentor method), 12
rejecting() (PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method), 43
rejecting_config (class in PrognosAIs.IO.Configs), 10
replace_root_path() (PrognosAIs.IO.LabelParser.LabelLoader method), 18
replace_root_path() (PrognosAIs.Model.Parsers.CallbackParser method), 38
resampling() (PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method), 43
resampling_config (class in PrognosAIs.IO.Configs), 10
reset_states() (PrognosAIs.Model.Metrics.MaskedSensitivity method), 36
reset_states() (PrognosAIs.Model.Metrics.MaskedSpecificity method), 36
reset_states() (PrognosAIs.Model.Metrics.Sensitivity method), 37
reset_states() (PrognosAIs.Model.Metrics.Specificity method), 37
ResNet (class in PrognosAIs.Model.Architectures.ResNet), 26
ResNet_18_2D (class in PrognosAIs.Model.Architectures.ResNet), 26
ResNet_18_3D (class in PrognosAIs.Model.Architectures.ResNet), 26
ResNet_18_multioutput_3D (class in PrognosAIs.Model.Architectures.ResNet), 26
ResNet_34_2D (class in PrognosAIs.Model.Architectures.ResNet), 26
ResNet_34_3D (class in PrognosAIs.Model.Architectures.ResNet), 27
result() (PrognosAIs.Model.Metrics.ConcordanceIndex method), 34
result() (PrognosAIs.Model.Metrics.DICE method), 34
result() (PrognosAIs.Model.Metrics.MaskedSensitivity method), 36
result() (PrognosAIs.Model.Metrics.MaskedSpecificity method), 36
result() (PrognosAIs.Model.Metrics.Sensitivity method), 37
result() (PrognosAIs.Model.Metrics.Specificity method), 37

S

saving() (PrognosAIs.Preprocessing.Preprocessors.SingleSamplePreprocessor method), 43
saving_config (class in PrognosAIs.IO.Configs), 10
Sensitivity (class in PrognosAIs.Model.Metrics), 37
set_precision_strategy() (PrognosAIs.Model.Trainer.Trainer method), 40
set_tf_config() (PrognosAIs.Model.Trainer.Trainer static method), 40
setup_augmentation() (PrognosAIs.IO.DataGenerator.HDF5Generator method), 16
setup_caching() (PrognosAIs.IO.DataGenerator.HDF5Generator method), 16
setup_caching_shuffling_steps() (PrognosAIs.IO.DataGenerator.HDF5Generator method), 16
setup_callbacks() (PrognosAIs.Model.Trainer.Trainer method), 40
setup_data_generator() (PrognosAIs.Model.Trainer.Trainer method), 41
setup_logger() (in module PrognosAIs.IO.utils), 19
setup_model() (PrognosAIs.Model.Trainer.Trainer method), 41
setup_sharding() (PrognosAIs.IO.DataGenerator.HDF5Generator method), 16
SingleSamplePreprocessor (class in PrognosAIs.Preprocessing.Preprocessors), 41
Specificity (class in PrognosAIs.Model.Metrics), 37
split_into_subsets() (PrognosAIs.Preprocessing.Preprocessors.BatchPreprocessor method), 41
StandardParser (class in PrognosAIs.Model.Parsers), 39
start() (PrognosAIs.Preprocessing.Preprocessors.BatchPreprocessor method), 41
start_local_pipeline() (PrognosAIs.Pipeline.Pipeline method), 55
start_slurm_pipeline() (PrognosAIs.Pipeline.Pipeline method), 55

T

THETA (PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_2D attribute), 22

THETA (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_121_3D attribute*), 22
THETA (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_2D attribute*), 23
THETA (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_169_3D attribute*), 23
THETA (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_2D attribute*), 23
THETA (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_201_3D attribute*), 24
THETA (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_2D attribute*), 24
THETA (*PrognosAIs.Model.Architectures.DenseNet.DenseNet_264_3D attribute*), 24
Timer (*class in PrognosAIs.Model.Callbacks*), 29
train_data_generator() (*PrognosAIs.Model.Trainer.Trainer property*), 41
train_model() (*PrognosAIs.Model.Trainer.Trainer method*), 41
Trainer (*class in PrognosAIs.Model.Trainer*), 39

U

Unet (*class in PrognosAIs.Model.Architectures.UNet*), 27
UNet_2D (*class in PrognosAIs.Model.Architectures.UNet*), 27
UNet_3D (*class in PrognosAIs.Model.Architectures.UNet*), 27
update_channel_size() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 47
update_labels() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 47
update_mask_size() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 47
update_metadata() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 47
update_output_channel_size() (*PrognosAIs.Preprocessing.Samples.ImageSample method*), 47
update_state() (*PrognosAIs.Model.Metrics.ConcordanceIndex method*), 34
update_state() (*PrognosAIs.Model.Metrics.DICE method*), 35
update_state() (*PrognosAIs.Model.Metrics.MaskedAUC method*), 35
update_state() (*PrognosAIs.Model.Metrics.MaskedCategoricalAccuracy method*), 35
update_state() (*PrognosAIs.Model.Metrics.MaskedSensitivity method*), 36
update_state() (*PrognosAIs.Model.Metrics.MaskedSpecificity method*), 36
update_state() (*PrognosAIs.Model.Metrics.Sensitivity method*), 37
update_state() (*PrognosAIs.Model.Metrics.Specificity method*), 37

V

validation_data_generator() (*PrognosAIs.Model.Trainer.Trainer property*), 41
VGG (*class in PrognosAIs.Model.Architectures.VGG*), 28
VGG_16_2D (*class in PrognosAIs.Model.Architectures.VGG*), 28
VGG_16_3D (*class in PrognosAIs.Model.Architectures.VGG*), 28
VGG_19_2D (*class in PrognosAIs.Model.Architectures.VGG*), 28
VGG_19_3D (*class in PrognosAIs.Model.Architectures.VGG*), 28

W

write_image_predictions_to_files() (*PrognosAIs.Model.Evaluators.Evaluator method*), 32
write_metrics_to_file() (*PrognosAIs.Model.Evaluators.Evaluator method*), 32
write_predictions_to_file() (*PrognosAIs.Model.Evaluators.Evaluator method*), 32