iNNvestigate Documentation

Release not set

Maximilian Alber

Mar 19, 2019

Contents:

This is the API documentation for the iNNvestigate library. It contains a references for the most important functions and classes of the library. For a usage guide and more information about the interplay of the componentes please visit the GitHub repo.

CHAPTER 1

API reference

1.1 innvestigate

innvestigate.analyzer.**create_analyzer**(*name*, *model*, ***kwargs*)
Instantiates the analyzer with the name 'name'

This convenience function takes an analyzer name creates the respective analyzer.

Alternatively analyzers can be created directly by instantiating the respective classes.

Parameters

- **name** – Name of the analyzer.
- **model** – The model to analyze, passed to the analyzer's __init__.
- **kwargs** – Additional parameters for the analyzer's .

Returns An instance of the chosen analyzer.

Raises **KeyError** – If there is no analyzer with the passed name.

1.2 innvestigate.analyzer

1.2.1 Interface and base code

exception innvestigate.analyzer.base.**NotAnalyzeableModelException**
Indicates that the model cannot be analyzed by an analyzer.

class innvestigate.analyzer.base.**AnalyzerBase**(*model*, *disable_model_checks=False*)
The basic interface of an iNNvestigate analyzer.

This class defines the basic interface for analyzers:

```
>>> model = create_keras_model()
>>> a = Analyzer(model)
>>> a.fit(X_train)  # If analyzer needs training.
>>> analysis = a.analyze(X_test)
>>>
>>> state = a.save()
>>> a_new = A.load(*state)
>>> analysis = a_new.analyze(X_test)
```

Parameters

- **model** – A Keras model.
- **disable_model_checks** – Do not execute model checks that enforce compatibility of analyzer and model.

Note: To develop a new analyzer derive from *AnalyzerNetworkBase*.

---

analyze(*X*)
Analyze the behavior of model on input *X*.

Parameters **X** – Input as expected by model.

fit(**args*, ***kwargs*)
Stub that eats arguments. If an analyzer needs training include *TrainerMixin*.

Parameters **disable_no_training_warning** – Do not warn if this function is called despite no training is needed.

fit_generator(**args*, ***kwargs*)
Stub that eats arguments. If an analyzer needs training include *TrainerMixin*.

Parameters **disable_no_training_warning** – Do not warn if this function is called despite no training is needed.

static load(*class_name*, *state*)
Resembles an analyzer from the state created by `analyzer.save()`.

Parameters

- **class_name** – The analyzer's class name.
- **state** – The analyzer's state.

static load_npz(*fname*)
Resembles an analyzer from the file created by `analyzer.save_npz()`.

Parameters **fname** – The file's name.

save()
Save state of analyzer, can be passed to `Analyzer.load()` to resemble the analyzer.

Returns The class name and the state.

save_npz(*fname*)
Save state of analyzer, can be passed to `Analyzer.load_npz()` to resemble the analyzer.

Parameters **fname** – The file's name.

class innvestigate.analyzer.base.**TrainerMixin**
Mixin for analyzer that adapt to data.

This convenience interface exposes a Keras like training routing to the user.

fit(*X=None*, *batch_size=32*, ***kwargs*)
Takes the same parameters as Keras's `model.fit()` function.

fit_generator(**args*, ***kwargs*)
Takes the same parameters as Keras's `model.fit_generator()` function.

class innvestigate.analyzer.base.**OneEpochTrainerMixin**
Exposes the same interface and functionality as *TrainerMixin* except that the training is limited to one epoch.

fit(**args*, ***kwargs*)
Same interface as *fit()* of *TrainerMixin* except that the parameter epoch is fixed to 1.

fit_generator(**args*, ***kwargs*)
Same interface as *fit_generator()* of *TrainerMixin* except that the parameter epoch is fixed to 1.

class innvestigate.analyzer.base.**AnalyzerNetworkBase**(*model*, *neuron_selection_mode='max_activation'*, *allow_lambda_layers=False*, ***kwargs*)
Convenience interface for analyzers.

This class provides helpful functionality to create analyzer's. Basically it:

- takes the input model and adds a layer that selects the desired output neuron to analyze.
- passes the new model to `_create_analysis()` which should return the analysis as Keras tensors.
- compiles the function and serves the output to *analyze()* calls.
- allows `_create_analysis()` to return tensors that are intercept for debugging purposes.

Parameters

- **neuron_selection_mode** – How to select the neuron to analyze. Possible values are 'max_activation', 'index' for the neuron (expects indices at *analyze()* calls), 'all' take all neurons.
- **allow_lambda_layers** – Allow the model to contain lambda layers.

analyze(*X*, *neuron_selection=None*)
Same interface as `Analyzer` besides

Parameters **neuron_selection** – If neuron_selection_mode is 'index' this should be an integer with the index for the chosen neuron.

create_analyzer_model()
Creates the analyze functionality. If not called beforehand it will be called by *analyze()*.

class innvestigate.analyzer.base.**ReverseAnalyzerBase**(*model*, *reverse_verbose=False*, *reverse_clip_values=False*, *reverse_project_bottleneck_layers=False*, *reverse_check_min_max_values=False*, *reverse_check_finite=False*, *reverse_keep_tensors=False*, *reverse_reapply_on_copied_layers=False*, ***kwargs*)
Convenience class for analyzers that revert the model's structure.

This class contains many helper functions around the graph reverse function *innvestigate.utils.keras.graph.reverse_model()*.

The deriving classes should specify how the graph should be reverted by implementing the following functions:

- `_reverse_mapping(layer)()` given a layer this function returns a reverse mapping for the layer as specified in *innvestigate.utils.keras.graph.reverse_model()* or None.

  This function can be implemented, but it is encouraged to implement a default mapping and add additional changes with the function `_add_conditional_reverse_mapping()` (see below).

  The default behavior is finding a conditional mapping (see below), if none is found, `_default_reverse_mapping()` is applied.

- `_default_reverse_mapping()` defines the default reverse mapping.
- `_head_mapping()` defines how the outputs of the model should be instantiated before the are passed to the reversed network.

Furthermore other parameters of the function *innvestigate.utils.keras.graph.reverse_model()* can be changed by setting the according parameters of the init function:

Parameters

- **reverse_verbose** – Print information on the reverse process.
- **reverse_clip_values** – Clip the values that are passed along the reverted network. Expects tuple (min, max).
- **reverse_project_bottleneck_layers** – Project the value range of bottleneck tensors in the reverse network into another range.
- **reverse_check_min_max_values** – Print the min/max values observed in each tensor along the reverse network whenever `analyze()` is called.
- **reverse_check_finite** – Check if values passed along the reverse network are finite.
- **reverse_keep_tensors** – Keeps the tensors created in the backward pass and stores them in the attribute `_reversed_tensors`.
- **reverse_reapply_on_copied_layers** – See *innvestigate.utils.keras.graph.reverse_model()*.

1.2.2 Gradient methods

class innvestigate.analyzer.gradient_based.**BaselineGradient**(*model*, *postprocess=None*, ***kwargs*)

Gradient analyzer based on build-in gradient.

Returns as analysis the function value with respect to the input. The gradient is computed via the build in function. Is mainly used for debugging purposes.

Parameters **model** – A Keras model.

class innvestigate.analyzer.gradient_based.**Gradient**(*model*, *postprocess=None*, ***kwargs*)

Gradient analyzer.

Returns as analysis the function value with respect to the input. The gradient is computed via the librarie's network reverting.

Parameters **model** – A Keras model.

class innvestigate.analyzer.gradient_based.**InputTimesGradient**(*model*, ***kwargs*)

Input*Gradient analyzer.

Parameters **model** – A Keras model.

class innvestigate.analyzer.gradient_based.**Deconvnet**(*model*, ***kwargs*)

Deconvnet analyzer.

Applies the "deconvnet" algorithm to analyze the model.

Parameters **model** – A Keras model.

class innvestigate.analyzer.gradient_based.**GuidedBackprop**(*model*, ***kwargs*)

Guided backprop analyzer.

Applies the "guided backprop" algorithm to analyze the model.

Parameters **model** – A Keras model.

class innvestigate.analyzer.gradient_based.**IntegratedGradients**(*model*, *steps=64*, ***kwargs*)

Integrated gradient analyzer.

Applies the "integrated gradient" algorithm to analyze the model.

Parameters

- **model** – A Keras model.
- **steps** – Number of steps to use average along integration path.

class innvestigate.analyzer.gradient_based.**SmoothGrad**(*model*, *augment_by_n=64*, ***kwargs*)

Smooth grad analyzer.

Applies the "smooth grad" algorithm to analyze the model.

Parameters

- **model** – A Keras model.
- **augment_by_n** – Number of distortions to average for smoothing.

1.2.3 Layer-wise relevance propagation

class innvestigate.analyzer.relevance_based.relevance_analyzer.**BaselineLRPZ**(*model*, ***kwargs*)

LRPZ analyzer - for testing purpose only.

Applies the "LRP-Z" algorithm to analyze the model. Based on the gradient times the input formula. **This formula holds only for ReLU/MaxPooling networks, for which LRP-Z collapses into the stated formula.**

Parameters **model** – A Keras model.

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRP**(*model*, **args*, ***kwargs*)

Base class for LRP-based model analyzers

Parameters

- **model** – A Keras model.

- **rule** – A rule can be a string or a Rule object, lists thereof or a list of conditions [(Condition, Rule), ...] gradient.
- **input_layer_rule** – either a Rule object, atuple of (low, high) the min/max pixel values of the inputs

create_rule_mapping(*layer*, *reverse_state*)

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPZ**(*model*, **args*, ***kwargs*)

LRP-analyzer that uses the LRP-Z rule

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPZIgnoreBias**(*model*, **args*, ***kwargs*)

LRP-analyzer that uses the LRP-Z-ignore-bias rule

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPEpsilon**(*model*, *epsilon=1e-07*, *bias=True*, **args*, ***kwargs*)

LRP-analyzer that uses the LRP-Epsilon rule

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPEpsilonIgnoreBias**(*model*, *epsilon=1*
07, **args*, ***kwargs*

LRP-analyzer that uses the LRP-Epsilon-ignore-bias rule

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPWSquare**(*model*, **args*, ***kwargs*)

LRP-analyzer that uses the DeepTaylor W**2 rule

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPFlat**(*model*, **args*, ***kwargs*)

LRP-analyzer that uses the LRP-Flat rule

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPAlphaBeta**(*model*, *alpha=None*, *beta=None*, *bias=True*, **args*, ***kwargs*)

Base class for LRP AlphaBeta

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPAlpha2Beta1**(*model*, **args*, ***kwargs*)

LRP-analyzer that uses the LRP-alpha-beta rule with a=2,b=1

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPAlpha2Beta1IgnoreBias**(*mod*
**ar*
***k*

LRP-analyzer that uses the LRP-alpha-beta-ignbias rule with a=2,b=1

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPAlpha1Beta0**(*model*, **args*, ***kwargs*)

LRP-analyzer that uses the LRP-alpha-beta rule with a=1,b=0

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPAlpha1Beta0IgnoreBias**(*mod*
**ar*
***k*

LRP-analyzer that uses the LRP-alpha-beta-ignbias rule with a=1,b=0

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPZPlus**(*model*, **args*, ***kwargs*)

LRP-analyzer that uses the LRP-alpha-beta rule with a=1,b=0

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPZPlusFast**(*model*, **args*, ***kwargs*)

The ZPlus rule is a special case of the AlphaBetaRule for alpha=1, beta=0 and assumes inputs x >= 0.

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPSequentialPresetA**(*model*, *ep-*
silon=0.1
**args*,
***kwargs*

Special LRP-configuration for ConvNets

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPSequentialPresetB**(*model*, *ep-*
silon=0.1
**args*,
***kwargs*

Special LRP-configuration for ConvNets

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPSequentialPresetAFlat**(*mod*
**ar*
***k*

Special LRP-configuration for ConvNets

class innvestigate.analyzer.relevance_based.relevance_analyzer.**LRPSequentialPresetBFlat**(*mod*
**ar*
***k*

Special LRP-configuration for ConvNets

1.2.4 Pattern methods

class innvestigate.analyzer.pattern_based.**PatternNet**(*model*, *patterns=None*, *pattern_type=None*, ***kwargs*)

PatternNet analyzer.

Applies the "PatternNet" algorithm to analyze the model's predictions.

Parameters

- **model** – A Keras model.
- **patterns** – Pattern computed by `innvestigate.tools.PatternComputer`. If None `fit()` needs to be called.
- **allow_lambda_layers** – Approximate lambda layers with the gradient.

- **reverse_project_bottleneck_layers** – Project the analysis vector into range [-1, +1]. (default: True)

class innvestigate.analyzer.pattern_based.**PatternAttribution**(*model*, *patterns=None*, *pattern_type=None*, ***kwargs*)

PatternAttribution analyzer.

Applies the "PatternNet" algorithm to analyze the model's predictions.

Parameters

- **model** – A Keras model.
- **patterns** – Pattern computed by innvestigate.tools.PatternComputer. If None fit() needs to be called.
- **allow_lambda_layers** – Approximate lambda layers with the gradient.
- **reverse_project_bottleneck_layers** – Project the analysis vector into range [-1, +1]. (default: True)

1.2.5 Deep Taylor

class innvestigate.analyzer.deeptaylor.**DeepTaylor**(*model*, **args*, ***kwargs*)

DeepTaylor for ReLU-networks with unbounded input

This class implements the DeepTaylor algorithm for neural networks with ReLU activation and unbounded input ranges.

Parameters **model** – A Keras model.

class innvestigate.analyzer.deeptaylor.**BoundedDeepTaylor**(*model*, *low=None*, *high=None*, ***kwargs*)

DeepTaylor for ReLU-networks with bounded input

This class implements the DeepTaylor algorithm for neural networks with ReLU activation and bounded input ranges.

Parameters

- **model** – A Keras model.
- **low** – Lowest value of the input range. See Z_B rule.
- **high** – Highest value of the input range. See Z_B rule.

1.2.6 DeepLIFT

class innvestigate.analyzer.deeplift.**DeepLIFTWrapper**(*model*, ***kwargs*)

Wrapper around DeepLIFT package

This class wraps the DeepLIFT package. For further explanation of the parameters check out: https://github.com/kundajelab/deeplift

Parameters

- **model** – A Keras model.
- **nonlinear_mode** – The nonlinear mode parameter.
- **reference_inputs** – The reference input used for DeepLIFT.

- **verbose** – Verbosity of the DeepLIFT package.

Note Requires the deeplift package.

analyze(*X*, *neuron_selection=None*)
Same interface as `Analyzer` besides

Parameters **neuron_selection** – If neuron_selection_mode is 'index' this should be an integer with the index for the chosen neuron.

1.2.7 Misc

class innvestigate.analyzer.misc.**Random**(*model*, *stddev=1*, ***kwargs*)
Returns noise.

Returns the Gaussian noise as analysis.

Parameters

- **model** – A Keras model.
- **stddev** – The standard deviation of the noise.

class innvestigate.analyzer.misc.**Input**(*model*, *neuron_selection_mode='max_activation'*, *allow_lambda_layers=False*, ***kwargs*)
Returns the input.

Returns the input as analysis.

Parameters **model** – A Keras model.

1.3 innvestigate.applications

1.3.1 Imagenet

Example applications for image classifcation.

Each function returns a pretrained ImageNet model. The models are based on keras.applications models and contain additionally pretrained patterns.

The returned dictionary contains the following keys: model, in, sm_out, out, image_shape, color_coding, preprocess_f, patterns.

Function parameters:

param load_weights Download or access cached weights.

param load_patterns Download or access cached patterns.

innvestigate.applications.imagenet.**vgg16**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**vgg19**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**resnet50**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**inception_v3**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**inception_resnet_v2**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**densenet121**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**densenet169**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**densenet201**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**nasnet_large**(*load_weights=False*, *load_patterns=False*)

innvestigate.applications.imagenet.**nasnet_mobile**(*load_weights=False*, *load_patterns=False*)

1.4 innvestigate.tools

1.4.1 Pattern Computation

class innvestigate.tools.pattern.**PatternComputer**(*model*, *pattern_type='linear'*, *compute_layers_in_parallel=True*, *gpus=None*)

Pattern computer.

Computes a pattern for each layer with a kernel of a given model.

Parameters

- **model** – A Keras model.
- **pattern_type** – A string or a tuple of strings. Valid types are 'linear', 'relu', 'relu.positive', 'relu.negative'.
- **compute_layers_in_parallel** – Not supported yet. Compute all patterns at once. Otherwise computer layer after layer.
- **gpus** – Not supported yet. Gpus to use.

compute(*X*, *batch_size=32*, *verbose=0*)

Compute and return the patterns for the model and the data *X*.

Parameters

- **X** – Data to compute patterns.
- **batch_size** – Batch size to use.
- **verbose** – As for keras model.fit.

compute_generator(*generator*, ***kwargs*)

Compute and return the patterns for the model and the data *X*.

Parameters

- **generator** – Data to compute patterns.
- **kwargs** – Same as for keras model.fit_generator.

1.4.2 Perturbation Analysis

class innvestigate.tools.perturbate.**Perturbation**(*perturbation_function*, *num_perturbed_regions=0*, *region_shape=(9, 9)*, *reduce_function=<function mean>*, *aggregation_function=<function mean>*, *pad_mode='reflect'*, *in_place=False*, *value_range=None*)

Perturbation of pixels based on analysis result.

Parameters

- **perturbation_function** (*function or callable or str*) – Defines the function with which the samples are perturbated. Can be a function or a string that defines a predefined perturbation function.
- **num_perturbed_regions** (*int*) – Number of regions to be perturbed.
- **reduce_function** (*function or callable*) – Function to reduce the analysis result to one channel, e.g. mean or max function.
- **aggregation_function** (*function or callable*) – Function to aggregate the analysis over subregions.
- **pad_mode** (*str or function or callable*) – How to pad if the image cannot be subdivided into an integer number of regions. As in numpy.pad.
- **in_place** (*bool*) – If true, the perturbations are performed in place, i.e. the input samples are modified.
- **value_range** (*tuple*) – Minimal and maximal value after perturbation as a tuple: (min_val, max_val). The input is clipped to this range

aggregate_regions(*analysis*)

static compute_perturbation_mask(*ranks*, *num_perturbated_regions*)

static compute_region_ordering(*aggregated_regions*)

expand_regions_to_pixels(*regions*)

pad(*analysis*)

perturbate_on_batch(*x*, *analysis*)

Parameters

- **x** (*numpy.ndarray*) – Batch of images.
- **analysis** (*numpy.ndarray*) – Analysis of this batch.

Returns Batch of perturbated images

Return type numpy.ndarray

perturbate_regions(*x*, *perturbation_mask_regions*)

reshape_region_pixels(*region_pixels*, *target_shape*)

reshape_to_regions(*analysis*)

class innvestigate.tools.perturbate.**PerturbationAnalysis**(*analyzer*, *model*, *generator*, *perturbation*, *steps=1*, *regions_per_step=1*, *recompute_analysis=False*, *verbose=False*)

Performs the perturbation analysis.

Parameters

- **analyzer** (innvestigate.analyzer.base.AnalyzerBase) – Analyzer.
- **model** (*keras.engine.training.Model*) – Trained Keras model.
- **generator** (innvestigate.utils.BatchSequence) – Data generator.
- **perturbation** (*innvestigate.tools.Perturbation*) – Instance of Perturbation class that performs the perturbation.
- **steps** (*int*) – Number of perturbation steps.
- **regions_per_step** (*float*) – Number of regions that are perturbed per step.
- **recompute_analysis** (*bool*) – If true, the analysis is recomputed after each perturbation step.
- **verbose** – If true, print some useful information, e.g. timing, progress etc.

compute_on_batch(*x*, *analysis=None*, *return_analysis=False*)

Computes the analysis and perturbes the input batch accordingly.

Parameters

- **x** (*numpy.ndarray*) – Samples.
- **analysis** – Analysis of x. If None, it is recomputed.

compute_perturbation_analysis()

evaluate_generator(*generator*, *steps=None*, *max_queue_size=10*, *workers=1*, *use_multiprocessing=False*)

Evaluates the model on a data generator.

The generator should return the same kind of data as accepted by *test_on_batch*. For documentation, refer to keras.engine.training.evaluate_generator (https://keras.io/models/model/)

evaluate_on_batch(*x*, *y*, *analysis=None*, *sample_weight=None*)

Perturbs the input batch and scores the model on the perturbed batch.

Parameters

- **x** (*numpy.ndarray*) – Samples.
- **y** (*numpy.ndarray*) – Labels.
- **analysis** (*numpy.ndarray*) – Analysis of x.
- **sample_weight** (*None*) – Sample weights.

Returns List of test scores.

Return type list

1.5 innvestigate.utils

innvestigate.utils.**model_wo_softmax**(**args*, ***kwargs*)

innvestigate.utils.**to_list**(*l*)
If not list, wraps parameter into a list.

class innvestigate.utils.**BatchSequence**(*Xs*, *batch_size=32*)
Batch sequence generator.

Take a (list of) input tensors and a batch size and creates a generators that creates a sequence of batches.

Parameters

- **Xs** – One or a list of tensors. First axis needs to have same length.
- **batch_size** – Batch size. Default 32.

class innvestigate.utils.**TargetAugmentedSequence**(*sequence*, *augment_f*)
Augments a sequence with a target on the fly.

Takes a sequence/generator and a function that creates on the fly for each batch a target. The generator takes a batch from that sequence, computes the target and returns both.

Parameters

- **sequence** – A sequence or generator.
- **augment_f** – Takes a batch and returns a target.

innvestigate.utils.**preprocess_images**(*images*, *color_coding=None*)
Image preprocessing

Takes a batch of images and: * Adjust the color axis to the Keras format. * Fixes the color coding.

Parameters

- **images** – Batch of images with 4 axes.
- **color_coding** – Determines the color coding. Can be None, 'RGBtoBGR' or 'BGRtoRGB'.

Returns The preprocessed batch.

innvestigate.utils.**postprocess_images**(*images*, *color_coding=None*, *channels_first=None*)
Image postprocessing

Takes a batch of images and reverts the preprocessing.

Parameters

- **images** – A batch of images with 4 axes.
- **color_coding** – The initial color coding, see *preprocess_images()*.
- **channels_first** – The output channel format.

Returns The postprocessed images.

1.5.1 Visualizations

innvestigate.utils.visualizations.**project**(*X*, *output_range=(0, 1)*, *absmax=None*, *input_is_positive_only=False*)
Projects a tensor into a value range.

Projects the tensor values into the specified range.

Parameters

- **X** – A tensor.
- **output_range** – The output value range.
- **absmax** – A tensor specifying the absmax used for normalizing. Default the absmax along the first axis.
- **input_is_positive_only** – Is the input value range only positive.

Returns The tensor with the values project into output range.

innvestigate.utils.visualizations.**heatmap**(*X*, *cmap_type='seismic'*, *reduce_op='sum'*, *reduce_axis=-1*, *alpha_cmap=False*, ***kwargs*)

Creates a heatmap/color map.

Create a heatmap or colormap out of the input tensor.

Parameters

- **X** – A image tensor with 4 axes.
- **cmap_type** – The color map to use. Default 'seismic'.
- **reduce_op** – Operation to reduce the color axis. Either 'sum' or 'absmax'.
- **reduce_axis** – Axis to reduce.
- **alpha_cmap** – Should the alpha component of the cmap be included.
- **kwargs** – Arguments passed on to *project()*

Returns The tensor as color-map.

innvestigate.utils.visualizations.**graymap**(*X*, ***kwargs*)

Same as *heatmap()* but uses a gray colormap.

innvestigate.utils.visualizations.**gamma**(*X*, *gamma=0.5*, *minamp=0*, *maxamp=None*)

Apply gamma correction to an input array X while maintaining the relative order of entries, also for negative vs positive values in X. the fxn firstly determines the max amplitude in both positive and negative direction and then applies gamma scaling to the positive and negative values of the array separately, according to the common amplitude.

Parameters

- **gamma** – the gamma parameter for gamma scaling
- **minamp** – the smallest absolute value to consider. if not given assumed to be zero (neutral value for relevance, min value for saliency, . . .). values above and below minamp are treated separately.
- **maxamp** – the largest absolute value to consider relative to the neutral value minamp if not given determined from the given data.

innvestigate.utils.visualizations.**clip_quantile**(*X*, *quantile=1*)

Clip the values of X into the given quantile.

1.5.2 Keras-Utils

innvestigate.utils.keras.graph.**copy_layer**(*layer*, *keep_bias=True*, *name_template=None*, *weights=None*, *reuse_symbolic_tensors=True*, ***kwargs*)

Copy a Keras layer

Copies a Keras layer.

Parameters

- **layer** – A layer that should be copied.
- **keep_bias** – Keep a potential bias.
- **weights** – Weights to set in the new layer. Options: np tensors, symbolic tensors, or None, in which case the weights from old_layers are used.
- **reuse_symbolic_tensors** – If the weights of the old_layer are used copy the symbolic ones or copy the Numpy weights.

Returns The new layer instance.

innvestigate.utils.keras.graph.**model_wo_softmax**(*model*)

Creates a new model w/o the final softmax activation.

innvestigate.utils.keras.graph.**get_model_execution_graph**(*model*, *keep_input_layers=False*)

Returns a dictionary representing the execution graph. Each key is the node's id as it is used by the reverse_model method.

Each associated value contains a dictionary with the following items:

- nid: the node id.
- layer: the layer creating this node.
- Xs: the input tensors (only valid if not in a nested container).
- Ys: the output tensors (only valid if not in a nested container).
- Xs_nids: the ids of the nodes creating the Xs.
- Ys_nids: the ids of nodes using the according output tensor.
- Xs_layers: the layer that created the accodring input tensor.
- Ys_layers: the layers using the according output tensor.

Parameters

- **model** – A kera model.
- **keep_input_layers** – Keep input layers.

innvestigate.utils.keras.graph.**reverse_model**(*model*, *reverse_mappings*, *default_reverse_mapping=None*, *head_mapping=None*, *stop_mapping_at_tensors=[]*, *verbose=False*, *return_all_reversed_tensors=False*, *clip_all_reversed_tensors=False*, *project_bottleneck_tensors=False*, *execution_trace=None*, *reapply_on_copied_layers=False*)

Reverses a Keras model based on the given reverse functions. It returns the reverted tensors for the according model inputs.

Parameters

- **model** – A Keras model.
- **reverse_mappings** – Either a callable that matches layers to mappings or a dictionary with layers as keys and mappings as values. Allowed as mapping forms are:
  - A function of form (A) f(Xs, Ys, reversed_Ys, reverse_state).
  - A function of form f(B) f(layer, reverse_state) that returns a function of form (A).
  - A `ReverseMappingBase` subclass.
- **default_reverse_mapping** – A function that reverses layers for which no mapping was given by param "reverse_mappings".
- **head_mapping** – Map output tensors to new values before passing them into the reverted network.
- **stop_mapping_at_tensors** – Tensors at which to stop the mapping. Similar to stop_gradient parameters for gradient computation.
- **verbose** – Print what's going on.
- **return_all_reversed_tensors** – Return all reverted tensors in addition to reverted model input tensors.
- **clip_all_reversed_tensors** – Clip each reverted tensor. False or tuple with min/max value.
- **project_bottleneck_tensors** – Project bottleneck layers in the reverting process into a given value range. False, True or (a, b) for projection range.
- **reapply_on_copied_layers** – When a model execution needs to linearized and copy layers before reapplying them. See `trace_model_execution()`.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index

F

G

H

I

L

M

N

O

P

R

S

T

V