As of now the implemented methods are:

* Package: Captum

Common Parameters:

(Additional, method specific parameters are listed later)

The inputs, baselines and target parameters are mostly common for all the methods.

inputs (*tensor or tuple of tensors*) – Input for which integrated gradients are computed. If forward_func takes a single tensor as input, a single input tensor should be provided. If forward_func takes multiple tensors as input, a tuple of the input tensors should be provided.

baselines (*scalar, tensor, tuple of scalars or tensors, optional*) –

Baselines define the starting point from which integral is computed and can be provided as:

- a single tensor, if inputs is a single tensor, with exactly the same dimensions as inputs or the first dimension is one and the remaining dimensions match with inputs.
- a single scalar, if inputs is a single tensor, which will be broadcasted for each input value in input tensor.
- a tuple of tensors or scalars, the baseline corresponding to each tensor in the inputs' tuple can be:
    - either a tensor with matching dimensions to corresponding tensor in the inputs' tuple or the first dimension is one and the remaining dimensions match with the corresponding input tensor.
    - or a scalar, corresponding to a tensor in the inputs' tuple. This scalar value is broadcasted for corresponding input tensor.

target (*int, tuple, tensor or list, optional*) – Output indices for which gradients are computed (for classification cases, this is usually the target class). If the network returns a scalar value per example, no target index is necessary

1. Saliency

Config Parameters:

- **abs** (*bool, optional*) – Returns absolute value of gradients if set to True, otherwise returns the (signed) gradients if False. Default: True

2. Integrated gradients

Config Parameters:

- **n_steps** (*int, optional*) – The number of steps used by the approximation method. Default: 50.
- **method** (*string, optional*) – Method for approximating the integral, one of *riemann_right*, *riemann_left*, *riemann_middle*, *riemann_trapezoid* or *gausslegendre*. Default: *gausslegendre* if no method is provided.
- **internal_batch_size** (*int, optional*) – Divides total #steps * #examples data points into chunks of size at most internal_batch_size, which are computed (forward / backward passes) sequentially. internal_batch_size must be at least equal to #examples. If internal_batch_size is None, then all evaluations are processed in one batch. Default: None
- **return_convergence_delta** (*bool, optional*) – Indicates whether to return convergence delta or not. If *return_convergence_delta* is set to True convergence delta will be returned in a tuple following attributions. Default: False

3. Feature ablation

Config Parameters:

- **feature_mask** (*tensor or tuple of tensors, optional*) – feature_mask defines a mask for the input, grouping features which should be ablated together. feature_mask should contain the same number of tensors as inputs. Each tensor should be the same size as the corresponding input or broadcastable to match the input tensor. Each tensor should contain integers in the range 0 to num_features - 1, and indices corresponding to the same feature should have the same value. Default: None
- **perturbations_per_eval** (*int, optional*) – Allows ablation of multiple features to be processed simultaneously in one call to forward_fn. Each forward pass will contain a maximum of perturbations_per_eval * #examples samples. Default: 1
- **show_progress** (*bool, optional*) – Displays the progress of computation. It will try to use tqdm if available for advanced features (e.g. time estimation). Otherwise, it will fallback to a simple output of progress. Default: False
- ****kwargs** (*Any, optional*) – Any additional arguments used by child classes of FeatureAblation (such as Occlusion) to construct ablations. These arguments are ignored when using FeatureAblation directly. Default: None

4. Guided Backpropagation

Config Parameters:

None

5. Deep lift

Config Parameters:

- **return_convergence_delta** (*bool, optional*) – Indicates whether to return convergence delta or not. If *return_convergence_delta* is set to True convergence delta will be returned in a tuple following attributions. Default: False
- **custom_attribution_func** (*callable, optional*) –

A custom function for computing final attribution scores. This function can take at least one and at most three arguments with the following signature:

  - custom_attribution_func(multipliers)
  - custom_attribution_func(multipliers, inputs)
  - custom_attribution_func(multipliers, inputs, baselines)

6. De-convolution

Config Parameters:

None

7. Guided grad-cam

Config Parameters:

- **interpolate_mode** (*str, optional*) – Method for interpolation, which must be a valid input interpolation mode for torch.nn.functional. These methods are “nearest”, “area”, “linear” (3D-only), “bilinear” (4D-only), “bicubic” (4D-only), “trilinear” (5D-only) based on the number of dimensions of the chosen layer output (which must also match the number of dimensions for the input tensor). Default: “nearest”
- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attribution with respect to the layer input or output in *LayerGradCam*. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to layer inputs, otherwise it will be computed with respect to layer outputs. Default: False

8. Layer activation

Config Parameters:

- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attribution with respect to the layer input or output. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to layer input, otherwise it will be computed with respect to layer output. Default: False

9. Layer conductance

Config Parameters:

- **n_steps** (*int, optional*) – The number of steps used by the approximation method. Default: 50.
- **method** (*string, optional*) – Method for approximating the integral, one of *riemann_right*, *riemann_left*, *riemann_middle*, *riemann_trapezoid* or *gausslegendre*. Default: *gausslegendre* if no method is provided.
- **internal_batch_size** (*int, optional*) – Divides total #steps * #examples data points into chunks of size at most internal_batch_size, which are computed (forward / backward passes) sequentially. internal_batch_size must be at least equal to 2 * #examples. Default: None
- **return_convergence_delta** (*bool, optional*) – Indicates whether to return convergence delta or not. If *return_convergence_delta* is set to True convergence delta will be returned in a tuple following attributions. Default: False
- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attribution with respect to the layer input or output. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to layer inputs, otherwise it will be computed with respect to layer outputs. Default: False

10. Layer grad shap

Config Parameters:

- **return_convergence_delta** (*bool, optional*) – Indicates whether to return convergence delta or not. If *return_convergence_delta* is set to True convergence delta will be returned in a tuple following attributions. Default: False
- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attribution with respect to the layer input or output. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to layer input, otherwise it will be computed with respect to layer output. Default: False

11. Gradient shap

Config Parameters:

- **return_convergence_delta** (*bool, optional*) – Indicates whether to return convergence delta or not. If *return_convergence_delta* is set to True convergence delta will be returned in a tuple following attributions. Default: False

12. Internal influence

Config Parameters:

- **n_steps** (*int, optional*) – The number of steps used by the approximation method. Default: 50.

- **method** (*string, optional*) – Method for approximating the integral, one of *riemann_right*, *riemann_left*, *riemann_middle*, *riemann_trapezoid* or *gausslegendre*. Default: *gausslegendre* if no method is provided.

- **internal_batch_size** (*int, optional*) – Divides total #steps * #examples data points into chunks of size at most internal_batch_size, which are computed (forward / backward passes) sequentially. internal_batch_size must be at least equal to #examples. Default: None

- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attribution with respect to the layer input or output. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to layer inputs, otherwise it will be computed with respect to layer outputs. Default: False

13. Input X Gradient

Config Parameters:

None

14. Deep lift shap

Config Parameters:

- **return_convergence_delta** (*bool, optional*) – Indicates whether to return convergence delta or not. If *return_convergence_delta* is set to True convergence delta will be returned in a tuple following attributions. Default: False

- **custom_attribution_func** (*callable, optional*) –

A custom function for computing final attribution scores. This function can take at least one and at most three arguments with the following signature:

  - custom_attribution_func(multipliers)
  - custom_attribution_func(multipliers, inputs)
  - custom_attribution_func(multipliers, inputs, baselines)

15. Layer Gradient X-Activation

Config Parameters:

- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attribution with respect to the layer input or output. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to layer input, otherwise it will be computed with respect to layer output. Default: False

16. Layer Deep lift

Config Parameters:

- **return_convergence_delta** (*bool, optional*) – Indicates whether to return convergence delta or not. If *return_convergence_delta* is set to True convergence delta will be returned in a tuple following attributions. Default: False
- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attribution with respect to the layer input or output. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to layer input, otherwise it will be computed with respect to layer output. Default: False
- **custom_attribution_func** (*callable, optional*) –

A custom function for computing final attribution scores. This function can take at least one and at most three arguments with the following signature:

  - custom_attribution_func(multipliers)
  - custom_attribution_func(multipliers, inputs)
  - custom_attribution_func(multipliers, inputs, baselines)

17. Layer Grad Cam

Config Parameters:

- **attribute_to_layer_input** (*bool, optional*) – Indicates whether to compute the attributions with respect to the layer input or output. If *attribute_to_layer_input* is set to True then the attributions will be computed with respect to the layer input, otherwise it will be computed with respect to layer output. Default: False
- **relu_attributions** (*bool, optional*) – Indicates whether to apply a ReLU operation on the final attribution, returning only non-negative attributions. Setting this flag to True matches the original GradCAM algorithm, otherwise, by default, both positive and negative attributions are returned. Default: False

18. Shapley Value Sampling

Config Parameters:

- **feature_mask** (*tensor or tuple of tensors, optional*) – feature_mask defines a mask for the input, grouping features which should be added together. feature_mask should contain the same number of tensors as inputs. Each tensor should be the same size as the corresponding input or broadcastable to match the input tensor. Values across all tensors should be integers in the range 0 to num_features - 1, and indices corresponding to the same feature should have the same value. Default: None
- **n_samples** (*int, optional*) – The number of feature permutations tested.
  Default: *25* if *n_samples* is not provided.
- **perturbations_per_eval** (*int, optional*) – Allows multiple ablations to be processed simultaneously in one call to forward_fn. Each forward pass will contain a maximum of perturbations_per_eval * #examples samples. Default: 1

- **show_progress** (*bool*, *optional*) – Displays the progress of computation. It will try to use tqdm if available for advanced features (e.g. time estimation). Otherwise, it will fallback to a simple output of progress. Default: False

`

Shapley value sampling takes lot of time for image input. Not recommended to perform on image data.

`

* CNN Visualization

1. Guided Backpropagation

During the backpropagation, the gradients of the outputs are computed with respect to the inputs (Springenberg et al., 2014). RELU activation function is applied to the input gradients, and direct backpropagation is performed, ensuring that the backpropagation of non-negative gradients does not occur.

2. Integrated Gradients

This technique (Sundararajan et al., 2017) calculates the path integral of the gradients along the straight-line path from the baseline x ′ to the input x.

3. Guided grad cam

The Gradient-weighted Class Activation Mapping approach provides a means to visualise the regions of an image input that are predominant in influencing the predictions made by the model.

4. Score cam

Activation maps are first extracted, and each activation then works as a mask on the original image, and its forward-passing score on the target class is obtained. Finally, the result can be generated by the linear combination of score-based weights and activation maps.

5. Vanilla Backpropagation

It is the standard gradient backpropagation technique through the deep neural network wherein the gradients are visualised at different layers and what is being learnt by the model is observed, given a random input image.

6. Grad Cam

The target output’s gradients are computed concerning the particularly given layer. The resultant gradients are averaged for each output channel (dimension 2 of output). The average gradient for each channel is then multiplied by the layer activations. The results are then added over all the channels. For the class feature weights w, global average pooling is performed over the feature maps A.

7. Grad Image Times

In this technique (Shrikumar et al., 2017b) the gradients are multiplied with the image itself.

8. Layer Activation Guided Backpropagation

It is similar to guided backpropagation, but instead of guiding the signal from the last layer and a specific target, it guides the signal from a specific layer and filter. The guided backpropagation method adds an additional guidance signal from the higher layers to the usual backpropagation.

9. Layer Visualization

It is the standard gradient backpropagation technique through the deep neural network wherein the gradients are visualised at different layers and what is being learnt by the model is observed, given a random input image.

10. Deep Dream

Given an arbitrary input image, any layer from the network can be picked, and the detected features at that layer can be enhanced. It is observed that the initial layers are sensitive to basic features in the input images, such as edges, and the deeper layers identify complex features from the input image.

* Torch Ray

Parameters:

The model, input and target parameters remain the same for all the methods.

- **model** (**torch.nn.Module**) – a model.
- **input** (**torch.Tensor**) – input tensor.
- **target** (int or **torch.Tensor**) – target label(s).

1. Excitation backpropagation

- **saliency_layer** (str or **torch.nn.Module**, optional) – name of the saliency layer (str) or the layer itself (**torch.nn.Module**) in the model at which to visualize. Default: **''** (visualize at input).
- **resize** (*bool or tuple, optional*) – if True, upsample saliency map to the same size as **input**. It is also possible to specify a pair (width, height) for a different size. Default: **False**.
- **resize_mode** (*str, optional*) – upsampling method to use. Default: **'bilinear'**.
- **smooth** (*float, optional*) – amount of Gaussian smoothing to apply to the saliency map. Default: **0**.
- **context_builder** (*type, optional*) – type of context to use. Default: **NullContext**.

- **gradient_to_saliency** (*function, optional*) – function that converts the pseudo-gradient signal to a saliency map. Default: **gradient_to_saliency()**.
- **get_backward_gradient** (*function, optional*) – function that generates gradient tensor to backpropagate. Default: **get_backward_gradient()**.
- **debug** (*bool, optional*) – if True, also return an **collections.OrderedDict** of **Probe** objects for all modules in the model. Default: **False**

2. Contrastive excitation backpropagation

Config Parameters:

- **saliency_layer** (str or **torch.nn.Module**) – name of the saliency layer (str) or the layer itself (**torch.nn.Module**) in the model at which to visualize.
- **contrast_layer** (str or **torch.nn.Module**) – name of the contrast layer (str) or the layer itself (**torch.nn.Module**).
- **classifier_layer** (str or **torch.nn.Module**, optional) – name of the last classifier layer (str) or the layer itself (**torch.nn.Module**). Defaults to **None**, in which case the functions tries to automatically identify the last layer. Default: **None**.
- **resize** (*bool or tuple, optional*) – If True resizes the saliency map to the same size as **input**. It is also possible to pass a (width, height) tuple to specify an arbitrary size. Default: **False**.
- **resize_mode** (*str, optional*) – Specify the resampling mode. Default: **'bilinear'**.
- **get_backward_gradient** (*function, optional*) – function that generates gradient tensor to backpropagate. Default: **common.get_backward_gradient()**.
- **debug** (*bool, optional*) – If True, also return **collections.OrderedDict** of **common.Probe** objects attached to all named modules in the model. Default: **False**.

3. Rise

Config Parameters:

- **seed** (*int, optional*) – manual seed used to generate random numbers. Default: **0**.
- **num_masks** (*int, optional*) – number of RISE random masks to use. Default: **8000**.
- **num_cells** (*int, optional*) – number of cells for one spatial dimension in low-res RISE random mask. Default: **7**.
- **filter_masks** (**torch.Tensor**, optional) – If given, use the provided pre-computed filter masks. Default: **None**.
- **batch_size** (*int, optional*) – batch size to use. Default: **128**.
- **p** (*float, optional*) – with prob p, a low-res cell is set to 0; otherwise, it’s 1. Default: **0.5**.
- **resize** (*bool or tuple of ints, optional*) – If True, resize saliency map to size of **input**. If False, don’t resize. If (width, height) tuple, resize to (width, height). Default: **False**.

- **resize_mode** (*str, optional*) – If resize is not None, use this mode for the resize function. Default: **'bilinear'**.

4. Deconv

Config parameters:

Same as Excitation Backpropagation

5. Grad Cam

Config parameters:

Same as Excitation Backpropagation

6. Gradient

Config parameters:

Same as Excitation Backpropagation

7. Guided Backpropagation

Config parameters:

Same as Excitation Backpropagation

8. Linear Approx.

Config parameters:

Same as Excitation Backpropagation

* Lucent

1. Render visualization

It is the PyTorch implementation of lucid for the explainability of deep learning models. It aims to explain the decision made by the deep neural network by explaining what is being learnt by the various layers of the network.

* Lime

1. Lime segmentation- Using Image explainer

Config Parameters:

- **image** – 3 dimension RGB image. If this is only two dimensional, we will assume it's a grayscale image and call gray2rgb.
- **classifier_fn** – classifier prediction probability function, which takes a numpy array and outputs prediction probabilities. For ScikitClassifiers , this is classifier.predict_proba.
- **labels** – iterable with labels to be explained.
- **hide_color** – TODO
- **top_labels** – if not None, ignore labels and produce explanations for the K labels with highest prediction probabilities, where K is this parameter.
- **num_features** – maximum number of features present in explanation
- **num_samples** – size of the neighborhood to learn the linear model