InterpretDL

Dec 15, 2022

Contents:

InterpretDL is an open source toolkit for interpretation algorithms based on PaddlePaddle. This toolkit contains three kinds of interpreters: input feature interpreters, intermediate-layer feature interpreters, and dataset-level interpreters. InterpretDL also provides the evaluation metrics to verify the trustworthiness of interpretation algorithms.

CHAPTER 1

Interpreters

1.1 Base Interpreter

1.1.1 Abstract Interpreter

class interpretdl.**Interpreter**(*model: callable*, *device: str*, ***kwargs*)

Interpreter is the base abstract class for all Interpreters. The implementation of any Interpreter should at least

(1) prepare `predict_fn` that outputs probability predictions, gradients or other desired intermediate results of the model, and

(2) implement the core function *`interpret()`* of the interpretation algorithm.

In general, we find this implementation is practical, makes the code more readable and can highlight the core function of the interpretation algorithm.

This kind of implementation works for all post-poc interpretation algorithms. While some algorithms may have different features and other fashions of implementations may be more suitable for them, our style of implementation can still work for most of them. So we follow this design for all Interpreters in this library.

Three sub-abstract Interpreters that implement *`_build_predict_fn()`* are currently provided in this file: *`InputGradientInterpreter`*, *`InputOutputInterpreter`*, *`IntermediateLayerInterpreter`*. For each of them, the implemented `predict_fn` can be used by several different algorithms. Therefore, the further implementations can focus on the core algorithm. More sub-abstract Interpreters will be provided if necessary.

> **Warning:** `use_cuda` would be deprecated soon. Use `device` directly.

Parameters

- **`model`** (*`callable`*) – A model with `forward()` and possibly `backward()` functions.
- **`device`** (*`str`*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

_build_predict_fn(***kwargs*)
Build `predict_fn` for interpreters. This will be called by *interpret()*.

_env_setup()
Prepare the environment setup. This is not always necessary because the setup can be done within the function of *_build_predict_fn()*.

interpret(***kwargs*)
Main function of the interpreter.

1.1.2 Sub-abstract: Input Gradient Interpreter

class interpretdl.**InputGradientInterpreter**(*model: callable*, *device: str*, ***kwargs*)
This is one of the sub-abstract Interpreters.

InputGradientInterpreter are used by input gradient based Interpreters. Interpreters that are derived from *InputGradientInterpreter* include *GradShapCVInterpreter*, *IntGradCVInterpreter*, *SmoothGradInterpreter*.

This Interpreter implements *_build_predict_fn()* that returns input gradient given an input.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

_build_predict_fn(*rebuild: bool = False*, *gradient_of: str = 'probability'*)
Build `predict_fn` for input gradients based algorithms. The model is supposed to be a classification model.

Parameters

- **rebuild** (*bool, optional*) – forces to rebuild. Defaults to `False`.
- **gradient_of** (*str, optional*) – computes the gradient of [`"loss"`, `"logit"` or `"probability"`] *w.r.t.* input data. Defaults to `"probability"`. Other options can get similar results while the absolute scale might be different.

1.1.3 Sub-abstract: Input Output Interpreter

class interpretdl.**InputOutputInterpreter**(*model: callable*, *device: str*, ***kwargs*)
This is one of the sub-abstract Interpreters.

InputOutputInterpreter are used by input-output correlation based Interpreters. Interpreters that are derived from *InputOutputInterpreter* include *OcclusionInterpreter*, *LIMECVInterpreter*, *SmoothGradInterpreter*.

This Interpreter implements *_build_predict_fn()* that returns the model's prediction given an input.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

_build_predict_fn(*rebuild: bool = False*, *output: str = 'probability'*)
Build `predict_fn` for Input-Output based algorithms. The model is supposed to be a classification model.

Parameters

- **rebuild** (*bool, optional*) – forces to rebuild. Defaults to `False`.
- **output** (*str, optional*) – computes the logit or probability. Defaults: `"probability"`. Other options can get similar results while the absolute scale might be different.

1.1.4 Sub-abstract: Intermediate-Layer Interpreter

class interpretdl.**IntermediateLayerInterpreter**(*model: callable*, *device: str*, ***kwargs*)

This is one of the sub-abstract Interpreters.

IntermediateLayerInterpreter exhibits features from intermediate layers to produce explanations. This interpreter extracts intermediate layers' features, but no gradients involved. Interpreters that are derived from *IntermediateLayerInterpreter* include *RolloutInterpreter*, *ScoreCAMInterpreter*.

This Interpreter implements *_build_predict_fn()* that returns the model's intermediate outputs given an input.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

_build_predict_fn(*rebuild: bool = False*, *target_layer: str = None*, *target_layer_pattern: str = None*)

Build `predict_fn` for IntermediateLayer based algorithms. The model is supposed to be a classification model. `target_layer` and `target_layer_pattern` cannot be set at the same time. See the arguments below.

Parameters

- **rebuild** (*bool, optional*) – forces to rebuild. Defaults to `False`.
- **target_layer** (*str, optional*) – the name of the desired layer whose features will output. This is used when there is only one layer to output. Conflict with `target_layer_pattern`. Defaults to `None`.
- **target_layer_pattern** (*str, optional*) – the pattern name of the layers whose features will output. This is used when there are several layers to output and they share a common pattern name. Conflict with `target_layer`. Defaults to `None`.

1.1.5 Sub-abstract: Transformer Interpreter

class interpretdl.**TransformerInterpreter**(*model: callable*, *device: str*, ***kwargs*)

This is one of the sub-abstract Interpreters.

`TransformerNLPInterpreter` are used by Interpreters for Transformer based model. Interpreters that are derived from `TransformerNLPInterpreter` include *BTNLPInterpreter*, *GANLPInterpreter*.

This Interpreter implements *_build_predict_fn()* that returns servral variables and gradients in each layer.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.

- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

_build_predict_fn(*rebuild: bool = False*, *embedding_name: str = None*, *attn_map_name: str = None*, *attn_v_name: str = None*, *attn_proj_name: str = None*, *gradient_of: str = None*)

Build `predict_fn` for transformer based algorithms. The model is supposed to be a classification model.

Parameters

- **rebuild** (*bool, optional*) – forces to rebuild. Defaults to `False`.
- **embedding_name** (*str, optional*) – the layer name for embedding, if in need.
- **attn_map_name** (*str, optional*) – the layer name for attention weights, if in need.
- **attn_v_name** (*str, optional*) – the layer name for attention value.
- **attn_proj_name** (*str, optional*) – the layer name for attention projection, if in need.
- **nlp** (*bool, default to False*) – whether the input data is for language test.

1.2 Input Feature Based Interpreters

1.2.1 Consensus

class interpretdl.**ConsensusInterpreter**(*InterpreterClass*, *list_of_models: list*, *device: str = 'gpu:0'*, ***kwargs*)

ConsensusInterpreter averages the explanations of a given Interpreter over a list of models. The averaged result is more like an explanation for the data, instead of specific models. For visual object recognition tasks, the Consensus explanation would be more aligned with the object than individual models.

More details regarding the Consensus method can be found in the original paper: https://arxiv.org/abs/2109.00707.

For reference, the `list_of_models` can be found from `paddle.vision.models` or PPClas.

Parameters

- **InterpreterClass** (*[type]*) – The given Interpreter defined in InterpretDL.
- **list_of_models** (*list*) – a list of trained models.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, ***kwargs*) → numpy.ndarray

The technical details are simple to understand for the Consensus method: Given the `inputs` and the interpretation algorithm (one of the Interpreters), each model in `list_of_models` will produce an explanation, then Consensus will concatenate all the explanations. Subsequent normalization and average can be done as users' preference. The suggested operation for input gradient based algorithms is average of the absolute values.

We leave the visualization to users. See the notebook example for an example.

```
import interpretdl as it
from paddle.vision.models import resnet34, resnet50, resnet101, mobilenet_v2
```

(continues on next page)

(continued from previous page)

```
list_models = {
    'resnet34': resnet34(pretrained=True),
    'resnet50': resnet50(pretrained=True),
    'resnet101': resnet101(pretrained=True),
    'mobilenet_v2': mobilenet_v2(pretrained=True)
}
consensus = ConsensusInterpreter(it.SmoothGradInterpreter, list_models.
↪values(), device='gpu:0')

import matplotlib.pyplot as plt
import numpy as np

cols = len(list_models) + 1
psize = 4
fig, ax = plt.subplots(1, cols, figsize=(cols*psize, 1*psize))

for axis in ax:
    axis.axis('off')

for i in range(len(list_models)):
    ax[i].imshow(np.abs(exp[i]).sum(0))
    ax[i].set_title(list(list_models.keys())[i])

ax[-1].imshow(np.abs(exp).sum(1).mean(0))
ax[-1].set_title('Consensus')
```

Parameters **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.

Returns Concatenated raw explanations.

Return type np.ndarray

1.2.2 Gradient Shap

class interpretdl.**GradShapCVInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Gradient SHAP Interpreter for CV tasks.

For input gradient based interpreters, the target issue is generally the vanilla input gradient's noises. The basic idea of reducing the noises is to use different similar inputs to get the input gradients and do the average.

GradShap uses noised inputs to get input gradients and then average.

More details regarding the GradShap method can be found in the original paper: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *labels: list = None*, *baselines: numpy.ndarray = None*, *n_samples: int = 5*, *noise_amount: float = 0.1*, *gradient_of: str = 'probability'*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*) → numpy.ndarray

The technical details of the GradShap method are described as follows: GradShap generates `n_samples` noised inputs, with the noise scale of `noise_amount`, and then computes the gradients *w.r.t.* these noised

inputs. A difference between `baselines` and noised inputs is considered. The final explanation is the multiplication between the gradients and the difference to `baselines`.

Parameters

- **inputs** (*str or list*) – The input image filepath or a list of filepaths or numpy array of read images.
- **labels** (*list or np.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **baselines** (*np.ndarray, optional*) – The baseline images to compare with. It should have the same shape as images and same length as the number of images. If None, the baselines of all zeros will be used. Default: `None`.
- **n_samples** (*int, optional*) – The number of randomly generated samples. Defaults to `5`.
- **noise_amount** (*float, optional*) – Noise level of added noise to each image. The std of Gaussian random noise is `noise_amount` * (x_{max} - x_{min}). Default: `0.1`.
- **gradient_of** (*str, optional*) – compute the gradient of [‘probability’, ‘logit’ or ‘loss’]. Default: `'probability'`. SmoothGrad uses probability for all tasks by default.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns the explanation result.

Return type np.ndarray

class interpretdl.**GradShapNLPInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

TODO: Inherit from a subabstract interpreter. Gradient SHAP Interpreter for NLP tasks.

For input gradient based interpreters, the target issue is generally the vanilla input gradient's noises. The basic idea of reducing the noises is to use different similar inputs to get the input gradients and do the average.

The inputs for NLP tasks are considered as the embedding features. So the noises or the changes of inputs are done for the embeddings.

More details regarding the GradShap method can be found in the original paper: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*data: tuple*, *labels: list = None*, *n_samples: int = 5*, *noise_amount: float = 0.1*, *embedding_name: str = 'word_embeddings'*, *return_pred: bool = True*) → numpy.ndarray

The technical details of the GradShap method for NLP tasks are similar for CV tasks, except the noises are added on the embeddings.

Parameters

- **data** (*tupleornp.ndarray*) – The inputs to the NLP model.
- **labels** (*listornp.ndarray, optional*) – The target label to analyze. If None, the most likely label will be used. Default: `None`.
- **n_samples** (*int, optional*) – The number of randomly generated samples. Defaults to `5`.
- **noise_amount** (*float, optional*) – Noise level of added noise to the embeddings. The std of Gaussian random noise is `noise_amount` * `embedding.mean()` * (x_{max} - x_{min}). Default: `0.1`.
- **embedding_name** (*str, optional*) – name of the embedding layer at which the noises will be applied. The name of embedding can be verified through `print(model)`. Defaults to `word_embeddings`.
- **return_pred** (*bool, optional*) – Whether or not to return predicted labels and probabilities. If True, a tuple of predicted labels, probabilities, and interpretations will be returned. There are useful for visualization. Else, only interpretations will be returned. Default: `True`.

Returns explanations, or (explanations, pred).

Return type np.ndarray or tuple

1.2.3 Integrated Gradients

class interpretdl.**IntGradCVInterpreter**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

Integrated Gradients Interpreter for CV tasks.

For input gradient based interpreters, the target issue is generally the vanilla input gradient's noises. The basic idea of reducing the noises is to use different similar inputs to get the input gradients and do the average.

IntGrad uses the Riemann approximation of the integral, i.e., interpolated values between a baseline (zero) and the original input as inputs, and computes the gradients which will be averaged as the final explanation.

More details regarding the Integrated Gradients method can be found in the original paper: https://arxiv.org/abs/1703.01365.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *labels: list = None*, *baselines: numpy.ndarray = None*, *steps: int = 50*, *num_random_trials: int = 10*, *gradient_of: str = 'probability'*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*) → numpy.ndarray

The technical details of the IntGrad method are described as follows: Given `inputs`, IntGrad interpolates `steps` points between `baselines` (usually set to zeros) and `inputs`. `baselines` can be set to `random`, so that `num_random_trials` baselines are used, instead of zeros. Then IntGrad computes the gradients *w.r.t.* these interpolated values and averages the results as final explanation.

Parameters

- **inputs** (*str or list*) – The input image filepath or a list of filepaths or numpy array of read images.

- **labels** (*list or tuple or np.ndarray or None, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **baselines** (*np.ndarray or None, optional*) – The baseline images to compare with. It should have the same shape as images and same length as the number of images. If None, the baselines of all zeros will be used. Default: `None`.
- **steps** (*int, optional*) – number of steps in the Riemann approximation of the integral. Default: `50`.
- **num_random_trials** (*int, optional*) – number of random initializations to take average in the end. Default: `10`.
- **gradient_of** (*str, optional*) – compute the gradient of [‘probability’, ‘logit’ or ‘loss’]. Default: `'probability'`. Multi-class classification uses probabitliy, while binary classification uses logit.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns the explanation result.

Return type np.ndarray

class interpretdl.**IntGradNLPInterpreter**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

Integrated Gradients Interpreter for NLP tasks.

For input gradient based interpreters, the target issue is generally the vanilla input gradient's noises. The basic idea of reducing the noises is to use different similar inputs to get the input gradients and do the average.

The inputs for NLP tasks are considered as the embedding features. So the noises or the changes of inputs are done for the embeddings.

More details regarding the Integrated Gradients method can be found in the original paper: https://arxiv.org/abs/1703.01365.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*raw_text: str*, *tokenizer: callable = None*, *text_to_input_fn: callable = None*, *label: list = None*, *steps: int = 50*, *gradient_of: str = 'logit'*, *embedding_name: str = 'word_embeddings'*, *max_seq_len: int = 128*, *visual: bool = False*) → numpy.ndarray

The technical details of the IntGrad method for NLP tasks are similar for CV tasks, except the noises are added on the embeddings.

Parameters

- **data** (*tupleornp.ndarray*) – The inputs to the NLP model.
- **labels** (*listornp.ndarray, optional*) – The target labels to analyze. If None, the most likely label will be used. Default: `None`.

- **steps** (*int, optional*) – number of steps in the Riemann approximation of the integral. Default: `50`.
- **gradient_of** (*str, optional*) – compute the gradient of ['probability', 'logit' or 'loss']. Default: `'logit'`. Multi-class classification uses probabitliy, while binary classification uses logit.
- **embedding_name** (*str, optional*) – name of the embedding layer at which the noises will be applied. The name of embedding can be verified through `print(model)`. Defaults to `word_embeddings`.

Returns explanations, or (explanations, pred).

Return type np.ndarray or tuple

1.2.4 LIME

class interpretdl.**LIMECVInterpreter**(*model: callable*, *device: str = 'gpu:0'*, *random_seed: int = None*)

LIME presents a locally explanation by fitting a set of perturbed samples near the target sample using an interpretable model, specifically a linear model.

The implementation is based on https://github.com/marcotcr/lime.

More details regarding the LIME method can be found in the original paper: https://arxiv.org/abs/1602.04938.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*data: str*, *interpret_class: int = None*, *top_k: int = 1*, *num_samples: int = 1000*, *batch_size: int = 50*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*)

Main function of the interpreter.

The implementation is based on https://github.com/marcotcr/lime.

Parameters

- **data** (*str*) – The input file path.
- **interpret_class** (*int, optional*) – The index of class to interpret. If None, the most likely label will be used. Default: `None`.
- **top_k** (*int, optional*) – Number of top classes to interpret. Will not be used if `interpret_class` is given. Default: `1`.
- **num_samples** (*int, optional*) – LIME sampling numbers. Larger number of samples usually gives more accurate interpretation. Default: `1000`.
- **batch_size** (*int, optional*) – Number of samples to forward each time. Default: `50`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.

- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: None.

Returns LIME results: {interpret_label_i: weights on features}

Return type [dict]

class interpretdl.**LIMENLPInterpreter**(*model: callable*, *device: str = 'gpu:0'*, *random_seed: int = None*)

LIME Interpreter for NLP tasks.

LIME presents a locally explanation by fitting a set of perturbed samples near the target sample using an interpretable model, specifically a linear model.

The implementation is based on https://github.com/marcotcr/lime.

More details regarding the LIME method can be found in the original paper: https://arxiv.org/abs/1602.04938.

Parameters

- **model** (*callable*) – A model with forward() and possibly backward() functions.
- **device** (*str*) – The device used for running model, options: "cpu", "gpu:0", "gpu:1" etc.
- **random_seed** (*int*) – random seed. Defaults to None.

interpret(*raw_text: str*, *tokenizer: callable = None*, *text_to_input_fn: callable = None*, *preprocess_fn: callable = None*, *unk_id: int = 0*, *pad_id: int = 0*, *classes_to_interpret: list = None*, *num_samples: int = 1000*, *batch_size: int = 50*, *max_seq_len: int = 128*, *visual: bool = False*)

Main function of the interpreter.

The implementation is based on https://github.com/marcotcr/lime.

Parameters

- **data** (*str*) – The raw string for analysis.
- **tokenizer** (*callable*) –
- **text_to_input** (*callable*) – A user-defined function that convert raw text string to a tuple of inputs that can be fed into the NLP model.
- **unk_id** (*int*) – The word id to replace occluded words. Typical choices include "", <unk>, and <pad>.
- **pad_id** (*int or None*) – The word id used to pad the sequences. If None, it means there is no padding. Default: None.
- **classes_to_interpret** (*list or numpy.ndarray, optional*) – The index of class to interpret. If None, the most likely label will be used. can be Default: None.
- **num_samples** (*int, optional*) – LIME sampling numbers. Larger number of samples usually gives more accurate interpretation. Default: 1000.
- **batch_size** (*int, optional*) – Number of samples to forward each time. Default: 50.
- **visual** (*bool, optional*) – Whether or not to visualize. Default: True.

Returns LIME results: {interpret_label_i: weights on features}

Return type [dict]

1.2.5 GLIME

class interpretdl.**GLIMECVInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

G-LIME CV Interpreter. This method integrates the global information from NormLIME or Average to the local explanation LIME.

More details can be found in this [pdf link](https://github.com/PaddlePaddle/InterpretDL/files/10110787/glime-aij-paper.pdf).

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

compute_global_weights(*g_name: str = 'normlime'*, *list_of_lime_explanations: list = None*, *list_file_paths: list = None*, *save_path: str = None*)

Compute the global weights, given the `list_of_lime_explanations`. This is done by NormLIME or Average Global Explanations, which are introduced in https://arxiv.org/abs/1909.04200 and https://arxiv.org/abs/1907.03039 respectively.

Parameters

- **g_name** (*str, optional*) – The method to aggregate local explanations. Defaults to `'normlime'`.
- **list_of_lime_explanations** (*list, optional*) – The LIME results. Defaults to None.
- **list_file_paths** (*list, optional*) – This is not implemented currently. Defaults to None.
- **save_path** (*str, optional*) – A path to save the global weights, which can be directly used the next time, and called by `set_global_weights()`. Defaults to None.

Raises `NotImplementedError` – NotImplementedError.

Returns Global Weights.

Return type dict

interpret(*data: str*, *interpret_class: int = None*, *top_k: int = 1*, *prior_method: str = 'none'*, *prior_reg_force: float = 1.0*, *num_samples: int = 1000*, *batch_size: int = 50*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*)

Note that for GLIME interpreter, *`set_global_weights()`* needs to be called before calling *`interpret()`*. Basically, the technical process of GLIME is similar to LIME. See the tutorial for more details.

Parameters

- **data** (*str*) – The input file path.
- **interpret_class** (*int, optional*) – The index of class to interpret. If None, the most likely label will be used. Default: `None`.
- **top_k** (*int, optional*) – Number of top classes to interpret. Will not be used if `interpret_class` is given. Default: `1`.
- **prior_method** – Prior method. Can be chosen from `{"none", "ridge"}`. Defaults to `"none"`, which is equivalent to LIME. If `none`, *`interpret()`* will use zeros as prior; Otherwise, the loaded prior will be used.

- **prior_reg_force** (*float, optional*) – The regularization force to apply. Default: `1.0`.
- **num_samples** (*int, optional*) – LIME sampling numbers. Larger number of samples usually gives more accurate interpretation. Default: `1000`.
- **batch_size** (*int, optional*) – Number of samples to forward each time. Default: 50
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*[type], optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The path to save the processed image. If None, the image will not be saved. Default: `None`.

Returns LIME results: {interpret_label_i: weights on features}

Return type [dict]

set_global_weights(*global_weights_info: str*)

Set directly the global weights without any pre-computations.

Parameters **global_weights_info** (*str or dict*) – A path of the file or the dict.

1.2.6 LIME With Global Prior

1.2.7 LRP

class interpretdl.**LRPCVInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Layer-wise Relevance Propagation (LRP) Interpreter for CV tasks.

The detailed introduction of LRP can be found in the tutorial. Layer-wise Relevance Propagation (LRP) is an explanation technique applicable to models structured as neural networks, where inputs can be e.g. images, videos, or text. LRP operates by propagating the prediction backwards in the neural network, by means of purposely designed local propagation rules.

Note that LRP requires `model` have `relprop()` and related implementations, see tutorial/assets/lrp_model. This is different from other interpreters, which do not have additional requirements for `model`.

More details regarding the LRP method can be found in the original paper: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0130140.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs*, *label=None*, *resize_to=224*, *crop_to=None*, *visual=True*, *save_path=None*)

The difficulty for LRP implementation does not reside the algorithm, but the model. The model should be implemented with `relprop()` functions, and the algorithm calls the relevance back-propagation, until the input layer, where the final explanation is obtained.

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **labels** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations/Relevance map for images.

Return type [numpy.ndarray]

1.2.8 Occlusion

class interpretdl.**OcclusionInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Occlusion Interpreter.

OcclusionInterpreter follows the simple idea of perturbation that says if the most important input features are perturbed, then the model's prediction will change the most. OcclusionInterpreter masks a block of pixels in the image, and then computes the prediction changes. According to the changes, the final explanation is obtained.

More details regarding the Occlusion method can be found in the original paper: https://arxiv.org/abs/1311.2901

Part of the code is modified from https://github.com/pytorch/captum/blob/master/captum/attr/_core/occlusion.py.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *sliding_window_shapes: tuple*, *labels: int = None*, *strides: int = 1*, *baselines: numpy.ndarray = None*, *perturbations_per_eval: int = 1*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*)

Part of the code is modified from https://github.com/pytorch/captum/blob/master/captum/attr/_core/occlusion.py.

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **sliding_window_shapes** (*tuple*) – Shape of sliding windows to occlude data.
- **labels** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: None

- **strides** (*int or tuple*) – The step by which the occlusion should be shifted by in each direction for each iteration. If int, the step size in each direction will be the same. Default: `1`.
- **baselines** (*numpy.ndarray or None, optional*) – The baseline images to compare with. It should have the same shape as images. If None, the baselines of all zeros will be used. Default: `None`.
- **perturbations_per_eval** (*int, optional*) – number of occlusions in each batch. Default: `1`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations for images

Return type [numpy.ndarray]

1.2.9 Smooth Gradients

class interpretdl.**SmoothGradInterpreter**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

Smooth Gradients Interpreter.

For input gradient based interpreters, the target issue is generally the vanilla input gradient's noises. The basic idea of reducing the noises is to use different similar inputs to get the input gradients and do the average.

Smooth Gradients method solves the problem of meaningless local variations in partial derivatives by adding random noise to the inputs multiple times and take the average of the gradients.

More details regarding the Smooth Gradients method can be found in the original paper: http://arxiv.org/abs/1706.03825.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *labels: list = None*, *noise_amount: int = 0.1*, *n_samples: int = 50*, *gradient_of: str = 'probability'*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*) → numpy.ndarray

The technical details of the SmoothGrad method are described as follows: SmoothGrad generates `n_samples` noised inputs, with the noise scale of `noise_amount`, and then computes the gradients *w.r.t.* these noised inputs. The final explanation is averaged gradients.

Parameters

- **inputs** (*str or list*) – The input image filepath or a list of filepaths or numpy array of read images.

- **labels** (*list or np.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: None.
- **noise_amount** (*int, optional*) – Noise level of added noise to the image. The std of Gaussian random noise is `noise_amount` * (x_{max} - x_{min}). Default: `0.1`.
- **n_samples** (*int, optional*) – The number of new images generated by adding noise. Default: `50`.
- **gradient_of** (*str, optional*) – compute the gradient of ['probability', 'logit' or 'loss']. Default: `'probability'`. SmoothGrad uses probability for all tasks by default.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns the explanation result.

Return type np.ndarray

1.2.10 Smooth Gradients V2

class interpretdl.**SmoothGradInterpreterV2**(*model: callable*, *device: str = 'gpu:0'*)

Smooth Gradients Interpreter.

For input gradient based interpreters, the target issue is generally the vanilla input gradient's noises. The basic idea of reducing the noises is to use different similar inputs to get the input gradients and do the average.

Smooth Gradients method solves the problem of meaningless local variations in partial derivatives by adding random noise to the inputs multiple times and take the average of the gradients.

This `SmoothGradInterpreterV2` only optimizes the GPU usage issue where large GPU memory usage may cause an error for large models and large batch sizes.

More details regarding the Smooth Gradients method can be found in the original paper: http://arxiv.org/abs/1706.03825.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *labels: list = None*, *noise_amount: int = 0.1*, *n_samples: int = 50*, *split: int = 2*, *gradient_of: str = 'probability'*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*) → numpy.ndarray

The technical details of the SmoothGrad method are described as follows: SmoothGrad generates `n_samples` noised inputs, with the noise scale of `noise_amount`, and then computes the gradients *w.r.t.* these noised inputs. The final explanation is averaged gradients. The difference to *SmoothGradInterpreter* is an additional argument `split`, where total samples are divided into `split` parts to pass the model, to avoid large GPU memory usages.

Parameters

- **inputs** (*str or list*) – The input image filepath or a list of filepaths or numpy array of read images.
- **labels** (*list or np.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **noise_amount** (*int, optional*) – Noise level of added noise to the image. The std of Gaussian random noise is `noise_amount` * (x_{max} - x_{min}). Default: `0.1`.
- **n_samples** (*int, optional*) – The number of new images generated by adding noise. Default: `50`.
- **split** (*int, optional*) – The number of splits. Default: `2`.
- **gradient_of** (*str, optional*) – compute the gradient of ['probability', 'logit' or 'loss']. Default: `'probability'`. SmoothGrad uses probability for all tasks by default.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns the explanation result.

Return type np.ndarray

1.2.11 NormLIME

class interpretdl.**NormLIMECVInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

NormLIME Interpreter for CV tasks.

(TODO) Some technical details will be complete soon.

More details regarding the NormLIME method can be found in the original paper: https://arxiv.org/abs/1909.04200.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*image_paths*, *num_samples=1000*, *batch_size=50*, *save_path='normlime_weights.npy'*, *temp_data_file='all_lime_weights.npz'*)

Main function of the interpreter.

(TODO) Some technical details will be complete soon.

Parameters

- **image_paths** (*list of strs*) – A list of image filepaths.

- **num_samples** (*int, optional*) – LIME sampling numbers. Larger number of samples usually gives more accurate interpretation. Default: 1000
- **batch_size** (*int, optional*) – Number of samples to forward each time. Default: 50
- **save_path** (*str, optional*) – The .npy path to save the normlime weights. It is a dictionary where the key is label and value is segmentation ids with their importance. Default: 'normlime_weights.npy'
- **temp_data_file** (*str, optional*) – The path to save the intermediate lime weights to avoid repeating computations. Default: 'all_lime_weights.npz'. Set to None will not save the intermediate lime weights.

Returns Global feature importance as a dict {label_i: weights on features}

Return type [dict] NormLIME weights

class interpretdl.**NormLIMENLPInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

NormLIME Interpreter for NLP tasks.

More details regarding the NormLIME method can be found in the original paper: https://arxiv.org/abs/1909.04200.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*list_of_raw_text*, *preprocess_fn: callable*, *num_samples: int*, *batch_size: int*, *unk_id: int = 0*, *pad_id: int = 0*, *lod_levels: int = None*, *save_path: str = 'normlime_weights.npy'*, *temp_data_file: str = 'all_lime_weights.npz'*)

NormLIMENLPInterpreter computes the LIME results of each sample of `data`, normalizes and averages the LIME results. `preprocess_fn` is used for coping with texts, see the tutorials for an example. `num_samples` and `batch_size` are LIME arguments, for the generated samples and the batch size of each pass.

Parameters

- **list_of_raw_text** (*str*) – The raw string for analysis.
- **preprocess_fn** (*Callable*) – A user-defined function that input raw string and outputs the a tuple of inputs to feed into the NLP model.
- **num_samples** (*int, optional*) – LIME sampling numbers. Larger number of samples usually gives more accurate interpretation. Default: 1000
- **batch_size** (*int, optional*) – Number of samples to forward each time. Default: 50
- **unk_id** (*int*) – The word id to replace occluded words. Typical choices include "", <unk>, and <pad>.
- **pad_id** (*int or None*) – The word id used to pad the sequences. If None, it means there is no padding. Default: None.
- **lod_levels** (*list or tuple or numpy.ndarray or None, optional*) – The lod levels for model inputs. It should have the length equal to number of outputs given by preprocess_fn. If None, lod levels are all zeros. Default: None.

- **save_path** (*str, optional*) – The .npy path to save the normlime weights. It is a dictionary where the key is label and value is segmentation ids with their importance. Default: 'normlime_weights.npy'
- **temp_data_file** (*str, optional*) – The .npz path to save the temporal LIME results, to avoid repeating the computations. Default: 'all_lime_weights.npz'

Returns {label_i: weights on features}

Return type [dict] NormLIME weights

1.2.12 TAM

class interpretdl.**TAMInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

TODO: Inherit from a subabstract interpreter. Transition Attention Maps Interpreter.

This is a specific interpreter for Transformers models. TAMInterpreter assumes that the information flowing inside the Transformer model follows the Markov Chain. Within this supposition, TAMInterpreter considers the attention maps as transition matrices and computes the explanation by multiplying the initial state with the attention maps, with integrated gradients.

More details regarding the Transition_Attention_Maps method can be found in the original paper: https://openreview.net/forum?id=TT-cf6QSDaQ.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *start_layer: int = 4*, *steps: int = 20*, *label: int = None*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*)

Given `inputs`, TAMInterpreter obtains all attention maps (of layers whose name matches `attention_layer_pattern`) and calculates their matrix multiplication. The `start_layer` controls the number of involved layers. The order of involving attention maps (from last layer to the first) is different from Rollout (from first to last). Then, an integrated gradients with `steps` is computed and multiplied to the attention result.

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **start_layer** (*int, optional*) – Compute the state from the start layer. Default: `4`.
- **steps** (*int, optional*) – number of steps in the Riemann approximation of the integral. Default: `50`.
- **labels** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.

- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations/heatmap for images

Return type [numpy.ndarray]

1.2.13 Generic Attention

class interpretdl.**GAInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Generic Attention Interpreter.

This is a specific interpreter for Bi-Modal Transformers models. GAInterpreter computes the attention map with gradients, and follows the operations that are similar to Rollout, with advanced modifications.

This implementation is suitable for models with self-attention in each modality, like CLIP.

More details regarding the Generic Attention method can be found in the original paper: https://arxiv.org/abs/2103.15679.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*image_input: str*, *text: str*, *text_tokenized: numpy.ndarray*, *vis_attn_layer_pattern: str = '^visual.transformer.resblocks.[0-9]*.attn.attn_map$'*, *txt_attn_layer_pattern: str = '^transformer.resblocks.[0-9]*.attn.attn_map$'*, *start_layer: int = 11*, *start_layer_text: int = 11*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*) → tuple

Given `image_input` and `text_tokenized`, GAInterpreter first obtains all attention maps (of layers whose name matches `vis_attn_layer_pattern` or `txt_attn_layer_pattern` for visual and text modalities respectively) and their gradients of the prediction. Then, GAInterpreter computes the multiplication between the attention map and its gradient for each block, and obtains the matrix multiplication of all blocks. The `start_layer` controls the number of involved layers. The order of involving attention maps (from the first layer to the last) is the same as Rollout (from first to last).

Parameters

- **image_input** (*str or np.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **text** (*str*) – original texts, for visualization.
- **text_tokenized** (*np.ndarray*) – The tokenized text for the model's input directly.
- **vis_attn_layer_pattern** (*str, optional*) – the pattern name of the layers whose features will output for visual modality. Defaults to `'^visual.transformer.resblocks.*.attn.attn_map$'`.
- **txt_attn_layer_pattern** (*str, optional*) – the pattern name of the layers whose features will output for text modality. Defaults to `'^transformer.resblocks.*.attn.attn_map$'`.
- **start_layer** (*int, optional*) – Compute the state from the start layer for visual modality. Defaults to `11`.
- **start_layer_text** (*int, optional*) – Compute the state from the start layer for text modality. Defaults to `11`.

- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns (text_relevance: np.ndarray, image_relevance: np.ndarray)

Return type tuple

class interpretdl.**GANLPInterpreter**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

Generic Attention Interpreter.

This is a specific interpreter for Bi-Modal Transformers models. GAInterpreter computes the attention map with gradients, and follows the operations that are similar to Rollout, with advanced modifications.

The following implementation is specially designed for Ernie.

More details regarding the Generic Attention method can be found in the original paper: https://arxiv.org/abs/2103.15679.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*raw_text: str*, *tokenizer: callable = None*, *text_to_input_fn: callable = None*, *label: int = None*, *start_layer: int = 11*, *attn_map_name='^[a-z]*.encoder.layers.[0-9]*.self_attn.attn_drop$'*, *gradient_of: str = 'logit'*, *max_seq_len=128*, *visual=False*)

Parameters

- **data** (*str or list of strs or numpy.ndarray*) – The input text filepath or a list of filepaths or numpy array of read texts.
- **start_layer** (*int, optional*) – Compute the state from the start layer. Default: `11`.
- **label** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of texts. If None, the most likely label for each text will be used. Default: `None`.
- **attn_map_name** (*str, optional*) – The layer name to obtain attention weights. Default: `^ernie.encoder.layers.*.self_attn.attn_drop$`.
- **gradient_of** (*str, optional*) – compute the gradient of ['probability', 'logit' or 'loss']. Default: `'logit'`. Multi-class classification uses probabitliy, while binary classification uses logit.

Returns interpretations for texts

Return type [numpy.ndarray]

class interpretdl.**GACVInterpreter**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

The following implementation is specially designed for Vision Transformer.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *start_layer: int = 3*, *attn_map_name='^blocks.[0-9]*.attn.attn_drop$'*, *label: int = None*, *gradient_of: str = 'probability'*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*)

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **start_layer** (*int, optional*) – Compute the state from the start layer. Default: `4`.
- **attn_map_name** (*str, optional*) – The layer name to obtain attention weights. Default: `^blocks.*.attn.attn_drop$`
- **label** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **gradient_of** (*str, optional*) – compute the gradient of ['probability', 'logit' or 'loss']. Default: `'probability'`. Multi-class classification uses probabitliy, while binary classification uses logit.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations/heatmap for images

Return type [numpy.ndarray]

1.2.14 Bidirectional Transformer Interpreter

class interpretdl.**BTCVInterpreter**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

Bidirectional Transformer Interpreter.

This is a specific interpreter for Transformers models, with two sub-processes: attentional perception, reasoning feedback.

The following implementation is specially designed for Vision Transformer.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *ap_mode: str = 'head'*, *start_layer: int = 3*, *steps: int = 20*, *attn_map_name='^blocks.[0-9]*.attn.attn_drop$'*, *attn_v_name='^blocks.[0-9]*.attn.qkv$'*, *attn_proj_name='^blocks.[0-9]*.attn.proj$'*, *gradient_of: str = 'probability'*, *label: int = None*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*)

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **ap_mode** (*str, optional*) – The approximation method of attentioanl perception stage, "head" for head-wise, "token" for token-wise. Default: `head`.
- **start_layer** (*int, optional*) – Compute the state from the start layer. Default: `4`.
- **steps** (*int, optional*) – number of steps in the Riemann approximation of the integral. Default: `20`.
- **attn_map_name** (*str, optional*) – The layer name to obtain the attention weights, head-wise/token-wise. Default: `^blocks.*.attn.attn_drop$`.
- **attn_v_name** (*str, optional*) – The layer name for query, key, value, token-wise. Default: `blocks.*.attn.qkv`.
- **attn_proj_name** (*str, optional*) – The layer name for linear projection, token-wise. Default: `blocks.*.attn.proj`.
- **gradient_of** (*str, optional*) – compute the gradient of ['probability', 'logit' or 'loss']. Default: `'probability'`. Multi-class classification uses probabitliy, while binary classification uses logit.
- **label** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations/heatmap for images

Return type [numpy.ndarray]

class interpretdl.**BTNLPInterpreter**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

Bidirectional Transformer Interpreter.

This is a specific interpreter for Transformers models, with two sub-processes: attentional perception, reasoning feedback.

The following implementation is specially designed for Ernie.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.

- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*raw_text: str*, *tokenizer: callable = None*, *text_to_input_fn: callable = None*, *label: list = None*, *ap_mode: str = 'head'*, *start_layer: int = 11*, *steps: int = 20*, *embedding_name='^[a-z]*.embeddings$'*, *attn_map_name='^[a-z]*.encoder.layers.[0-9]*.self_attn.attn_drop$'*, *attn_v_name='^[a-z]*.encoder.layers.[0-9]*.self_attn.v_proj$'*, *attn_proj_name='^[a-z]*.encoder.layers.[0-9]*.self_attn.out_proj$'*, *gradient_of: str = 'logit'*, *max_seq_len=128*, *visual=False*)

Parameters

- **data** (*str or list of strs or numpy.ndarray*) – The input text filepath or a list of filepaths or numpy array of read texts.
- **ap_mode** (*str, default to head-wise*) – The approximation method of attentioanl perception stage, "head" for head-wise, "token" for token-wise. Default: `head`.
- **start_layer** (*int, optional*) – Compute the state from the start layer. Default: `11`.
- **steps** (*int, optional*) – number of steps in the Riemann approximation of the integral. Default: `20`.
- **embedding_name** (*str, optional*) – The layer name for embedding, head-wise/token-wise. Default: `^ernie.embeddings$`.
- **attn_map_name** (*str, optional*) – The layer name to obtain the attention weights, head-wise/token-wise. Default: `^ernie.encoder.layers.*.self_attn.attn_drop$`.
- **attn_v_name** (*str, optional*) – The layer name for value projection, token-wise. Default: `^ernie.encoder.layers.*.self_attn.v_proj$`.
- **attn_proj_name** (*str, optional*) – The layer name for linear projection, token-wise. Default: `ernie.encoder.layers.*.self_attn.out_proj$`.
- **gradient_of** (*str, optional*) – compute the gradient of ['probability', 'logit' or 'loss']. Default: `'logit'`. Multi-class classification uses probabitliy, while binary classification uses logit.
- **label** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of texts. If None, the most likely label for each text will be used. Default: `None`.

Returns interpretations for texts

Return type [numpy.ndarray]

1.3 Intermediate-Layer Feature Interpreters

1.3.1 Grad-CAM

class interpretdl.**GradCAMInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Gradient CAM Interpreter.

Given a convolutional network and an image classification task, classification activation map (CAM) can be derived from the global average pooling and the last fully-connected layer, and show the important regions that affect model's decisions.

GradCAM further looks at the gradients flowing into one of the convolutional layers to give weight to activation maps. Note that if there is a global average pooling layer in the network, GradCAM targeting the last layer is equivalent to CAM.

More details regarding the CAM method can be found in the original paper: https://arxiv.org/abs/1512.04150.

More details regarding the GradCAM method can be found in the original paper: https://arxiv.org/abs/1610.02391.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret (*inputs: str*, *target_layer_name: str*, *label: list = None*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*) → numpy.ndarray

The technical details of the GradCAM method are described as follows: GradCAM computes the feature map at the layer of `target_layer_name` and the gradient of the objective function *w.r.t.* `target_layer_name`. With the average of gradients along the spatial dimensions, gradients will be multiplied with feature map, following by a ReLU activation to produce the final explanation.

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **target_layer_name** (*str*) – The target layer to calculate gradients.
- **labels** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations/heatmap for images

Return type [numpy.ndarray]

1.3.2 Score CAM

class interpretdl.**ScoreCAMInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Score-CAM Interpreter.

ScoreCAMInterpreter bridges the gap between perturbation-based and CAM-based methods, and derives the weight of activation maps in an intuitively understandable way.

More details regarding the Score CAM method can be found in the original paper: https://arxiv.org/abs/1910.01279.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *target_layer_name: str*, *labels: list = None*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*) → numpy.ndarray

Main function of the interpreter.

(TODO) The technical details will be described later.

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **target_layer_name** (*str*) – The target layer to calculate gradients.
- **labels** (*list or tuple or numpy.ndarray, optional*) – The target labels to analyze. The number of labels should be equal to the number of images. If None, the most likely label for each image will be used. Default: `None`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations/heatmap for images

Return type [np.ndarray]

1.3.3 Rollout

class interpretdl.**RolloutInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Rollout Interpreter.

This is a specific interpreter for Transformers models. RolloutInterpreter assumes that attentions can be linearly combined and the obtained score is able to show the scores of tokens, which gives an explanation of token importance.

More details regarding the Rollout method can be found in the original paper: https://arxiv.org/abs/2005.00928.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*inputs: str*, *start_layer: int = 0*, *attention_layer_pattern: str = '^blocks.*.attn.attn_drop$'*, *resize_to: int = 224*, *crop_to: int = None*, *visual: bool = True*, *save_path: str = None*)

Given `inputs`, RolloutInterpreter obtains all attention maps (of layers whose name matches `attention_layer_pattern`) and calculates their matrix multiplication. The `start_layer` controls the number of involved layers.

Parameters

- **inputs** (*str or list of strs or numpy.ndarray*) – The input image filepath or a list of filepaths or numpy array of read images.
- **start_layer** (*int, optional*) – The index of the start layer involving the computation of attentions. Defaults to `0`.
- **attention_layer_pattern** (*str, optional*) – the string pattern to pick the layers that match the pattern. Defaults to `^blocks.*.attn.attn_drop$`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **visual** (*bool, optional*) – Whether or not to visualize the processed image. Default: `True`.
- **save_path** (*str, optional*) – The filepath(s) to save the processed image(s). If None, the image will not be saved. Default: `None`.

Returns interpretations/heatmap for images.

Return type [np.ndarray]

1.4 Dataset-Level Interpreters

1.4.1 Training Dynamics

class interpretdl.**TrainingDynamics**(*paddle_model: callable*, *device: str = 'gpu:0'*)

Training Dynamics Interpreter focus the behavior of each training sample by running a normal SGD training process.

By recording the training dynamics, interpreter can diagnose dataset with hand-designed features or by learning solution.

After interpretation on the level of data, we can handle better the datasets with underlying label noises, thus can achieve a better performance on it.

More training dynamics based methods including [Forgetting Events] and [Dataset Mapping] will be available in this interpreter.

Parameters

- **paddle_model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `paddle_model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

generator(*train_loader: callable*, *optimizer: <module 'paddle.optimizer' from '/home/docs/checkouts/readthedocs.org/user_builds/interpretdl/envs/latest/lib/python3.7/site-packages/paddle/optimizer/__init__.py'>*, *epochs: int*)

Run the training process and record the forgetting events statistics.

Parameters

- **train_loader** (*callable*) – A training data generator.
- **optimizer** (*paddle.optimizer*) – The paddle optimizer.
- **epochs** (*int*) – The number of epochs to train the model.

Returns A pointwise training dynamics(history) for each epoch.

Return type training_dynamics (dict)

save(*logits*, *assigned_targets*, *label_flip=None*, *save_path=None*)
Save transformed training dynamics .

Parameters **save_path** (*_type_, optional*) – The filepath to save the processed image. If None, the image will not be saved. Default: None

transform(*logits*, *assigned_targets*)
Transform training dynamics with linear interpolation.

Parameters

- **logits** (*dict*) – A dictionary recording training dynamics.
- **assigned_targets** (*list*) – The assigned targets of dataset,.

1.4.2 Beyond Hand-designed Feature Interpreter

class interpretdl.**BHDFInterpreter**(*detector: callable = None*, *device: str = 'gpu:0'*)
BHDFInterpreter takes the training dynamics as raw input and lets an LSTM model to predict whether the sample is mislabeled or clean. We have provided a pre-trained LSTM, which can be directly used for identifying the mislabels.

TODO: to add the arxiv link. More details regarding this method can be found in the original paper: [link_to_be_annonced]().

For reproduction experiments, refer to [this repo](https://github.com/Christophe-Jia/mislabel-detection).

Parameters

- **detector** (*callable, optional*) – A detector model for identifying the mislabeled samples. Defaults to None.
- **device** (*str, optional*) – The device used for running `detector`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc. Defaults to 'gpu:0'.

interpret(*training_dynamics=None*, *training_dynamics_path='assets/training_dynamics.npz'*)
Call this function to rank samples' correctness.

Parameters

- **training_dynamics** (*dict, optional*) – Training dynamics is a dictionary, which has keys as follows:
- **{** –
- **label_flip** – list: The position of label contamination where True indicates label noise;
- **labels** – numpy.ndarray: with shape of length of dataset * class number, generated by `TrainingDynamics.generator`;
- **td** – numpy.ndarray: with shape of length of dataset * training epochs * class number, point-wise probability for each epoch, generated by `TrainingDynamics.generator`.
- **}** –

Returns (order,predictions) where order is {ranking of label correctness in form of data indices list} and predictions is {point-wise predictions as clean}.

Return type (numpy.ndarray, list)

1.4.3 Forgetting Events

class interpretdl.**ForgettingEventsInterpreter**(*model: callable*, *device: str = 'gpu:0'*)

Forgetting Events Interpreter computes the frequency of forgetting events for each training sample by running a normal training process. The training sample undergoes a forgetting event if it is misclassified at step t+1 after having been correctly classified at step t.

A training sample would be more probable to be mislabeled or hard to learn if it has more forgetting events happened.

More details regarding the Forgetting Events method can be found in the original paper: https://arxiv.org/abs/1812.05159.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc.

interpret(*train_reader: callable*, *optimizer: <module 'paddle.optimizer' from '/home/docs/checkouts/readthedocs.org/user_builds/interpretdl/envs/latest/lib/python3.7/site-packages/paddle/optimizer/__init__.py'>*, *batch_size: int*, *epochs: int*, *find_noisy_labels=False*, *save_path=None*)

Run the training process and record the forgetting events statistics.

Parameters

- **train_reader** (*callable*) – A training data generator.
- **optimizer** (*paddle.optimizer*) – The paddle optimizer.
- **batch_size** (*int*) – Number of samples to forward each time.
- **epochs** (*int*) – The number of epochs to train the model.
- **find_noisy_labels** (*bool, optional*) – whether to find noisy labels. Defaults to False.
- **save_path** (*_type_, optional*) – The filepath to save the processed image. If None, the image will not be saved. Default: None

Returns (count_forgotten, forgotten) where count_forgotten is {count of forgetting events: list of data indices with such count of forgetting events} and forgotten is {data index: numpy.ndarray of wrong predictions that follow true predictions in the training process}.

Return type (dict, dict)

1.4.4 SGDNoise

See Tutorials.

1.4.5 TrainIng Data analYzer (TIDY)

See Tutorials.

CHAPTER 2

Interpreter Trustworthiness Evaluation Metrics

2.1 Abstract Evaluator

class interpretdl.**InterpreterEvaluator**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

InterpreterEvaluator is the base abstract class for all interpreter evaluators. The core function `evaluate` should be implemented.

All evaluators aim to evaluate the trustworthiness of the interpretation algorithms. Besides theoretical verification of the algorithm, here the evaluators validate the trustworthiness by looking through the obtained explanations from the interpretation algorithms. Different evaluators are provided.

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions. This is not always required if the model is not involved.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc. Again, this is not always required if the model is not involved.

2.2 DeletionInsertion

class interpretdl.**DeletionInsertion**(*model: callable*, *device: str*, *compute_deletion: bool = True*, *compute_insertion: bool = True*, ***kwargs*)

Deletion & Insertion Interpreter Evaluation method.

The evaluation of interpretation algorithms follows the intuition that flipping the most salient pixels first should lead to high performance decay. Perturbation-based examples can therefore be used for the trustworthiness evaluations of interpretation algorithms.

The Deletion metric is computed as follows. The perturbation starts from an original image, perturbs (zeros out) the most important pixels in the input, and then computes the responses of the trained model. So that a curve, with ratios of perturbed pixels as x-axis and probabilities as y-axis, can be obtained and the area under this curve is the deletion score.

The Insertion metric is similar, but the perturbation starts from a zero image, inserts the most important pixels to the input, and then computes the responses of the trained model. A similar curve can be obtained and the area under this curve is the insertion score.

More details regarding the Deletion & Insertion method can be found in the original paper: https://arxiv.org/abs/1806.07421

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions. This is not always required if the model is not involved.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc. Again, this is not always required if the model is not involved.
- **compute_deletion** (*bool, optional*) – Whether compute deletion score. Defaults to `True`.
- **compute_insertion** (*bool, optional*) – Whether compute insertion score. Defaults to `True`.

Raises `ValueError` – At least one of `compute_deletion` and `compute_insertion` must be True.

evaluate(*img_path: str*, *explanation: dict*, *batch_size: int = None*, *resize_to: int = 224*, *crop_to: int = None*, *limit_number_generated_samples: int = None*) → dict

Given `img_path`, DeletionInsertion first generates perturbed samples of deletion and insertion, respectively, according to the order provided by `explanation`. The number of samples is defined by `limit_number_generated_samples` (a sampling is used for the numbers are different). Then DeletionInsertion computes the probabilities of these perturbed samples, and the mean of all probabilities of the class of interest is computed for the final score.

Note that LIME produces explanations based on superpixels, the number of perturbed samples is originally equal to the number of superpixels. So if `limit_number_generated_samples` is None, then the number of superpixels is used. For other explanations that produce the explanation of the same spatial dimension as the input image, `limit_number_generated_samples` is set to `20` if not given.

Parameters

- **img_path** (*str*) – a string for image path.
- **explanation** (*dict or np.ndarray*) – the explanation result from an interpretation algorithm.
- **batch_size** (*int or None, optional*) – batch size for each pass. Defaults to `None`.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.
- **limit_number_generated_samples** (*int or None, optional*) – a maximum value for samples of perturbation. If None, it will be automatically chosen. The number of superpixels is used for LIME explanations, otherwise, `20` is to be set. Defaults to `None`.

Returns A dict containing `'deletion_score'`, `'del_probas'`, `'deletion_images'`, `'insertion_score'`, `'ins_probas'` and `'insertion_images'`, if `compute_deletion` and `compute_insertion` are both True.

Return type dict

2.3 Perturbation

class interpretdl.**Perturbation**(*model: callable*, *device: str = 'gpu:0'*, *compute_MoRF: bool = True*, *compute_LeRF: bool = True*, ***kwargs*)

Perturbation based Evaluations.

The evaluation of interpretation algorithms follows the intuition that flipping the most salient pixels first should lead to high performance decay. Perturbation-based examples can therefore be used for the trustworthiness evaluations of interpretation algorithms.

Two metrics are provided: most relevant first (MoRF) and least relevant first (LeRF).

The MoRF metric is computed as follows. The perturbation starts from an original image, perturbs (zeros out) the most important pixels in the input, and then computes the responses of the trained model. So that a curve, with ratios of perturbed pixels as x-axis and probabilities as y-axis, can be obtained and the area under this curve is the MoRF score.

The LeRF metric is similar, but the perturbation perturbs (zeros out) the least important pixels in the input and then computes the responses of the trained model. A similar curve can be obtained and the area under this curve is the LeRF score.

Note that MoRF is equivalent to Deletion, but LeRF is NOT equivalent to Insertion.

More details of MoRF and LeRF can be found in the original paper: https://arxiv.org/abs/1509.06321.

summary

Parameters

- **model** (*callable*) – A model with `forward()` and possibly `backward()` functions. This is not always required if the model is not involved.
- **device** (*str*) – The device used for running `model`, options: `"cpu"`, `"gpu:0"`, `"gpu:1"` etc. Again, this is not always required if the model is not involved.
- **compute_MoRF** (*bool, optional*) – Whether comptue MoRF score. Defaults to True.
- **compute_LeRF** (*bool, optional*) – Whether comptue LeRF score. Defaults to True.

Raises `ValueError` – 'At least one of `compute_MoRF` and `compute_LeRF` must be True.'

evaluate(*img_path: str*, *explanation: list*, *batch_size=None*, *resize_to=224*, *crop_to=None*, *limit_number_generated_samples=None*) → dict

Given `img_path`, Perturbation first generates perturbed samples of MoRF and LeRF respectively, according to the order provided by `explanation`. The number of samples is defined by `limit_number_generated_samples` (a sampling is used for the numbers are different). Then Perturbation computes the probabilities of these perturbed samples, and the mean of all probabilities of the class of interest is computed for the final score.

Note that LIME produces explanations based on superpixels, the number of perturbed samples is originally equal to the number of superpixels. So if `limit_number_generated_samples` is None, then the number of superpixels is used. For other explanations that produce the explanation of the same spatial dimension as the input image, `limit_number_generated_samples` is set to `20` if not given.

Parameters

- **img_path** (*str*) – a string for image path.

- **explanation** (*list or np.ndarray*) – the explanation result from an interpretation algorithm.
- **batch_size** (*int or None, optional*) – batch size for each pass. Defaults to None.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being resize_to. Defaults to 224.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size crop_to. If None, no crop will be performed. Defaults to None.
- **limit_number_generated_samples** (*int or None, optional*) – a maximum value for samples of perturbation. If None, it will be automatically chosen. The number of superpixels is used for LIME explanations, otherwise, 20 is to be set. Defaults to None.

Returns A dict containing 'MoRF_score', 'MoRF_probas', 'MoRF_images', 'LeRF_score', 'LeRF_probas' and 'LeRF_images', if compute_MoRF and compute_LeRF are both True.

Return type dict

2.4 Infidelity

class interpretdl.**Infidelity**(*model: callable*, *device: str = 'gpu:0'*, ***kwargs*)

Infidelity Interpreter Evaluation method.

The idea of fidelity is similar to the faithfulness evaluation, to evaluate how faithful/reliable/loyal of the explanations to the model. (In)fidelity measures the normalized squared Euclidean distance between two terms: the product of a perturbation and the explanation, and the difference between the model's response to the original input and the one to the perturbed input, i.e.

$$INFD(\Phi, f, x) = \mathbb{E}_{I \sim \mu_I}[(I^T \Phi(f, x) - (f(x) - f(x - I)))^2],$$

where the meaning of the symbols can be found in the original paper.

A normalization is added, which is not in the paper but in the official implementation:

$$\beta = \frac{\mathbb{E}_{I \sim \mu_I}[I^T \Phi(f, x)(f(x) - f(x - I))]}{\mathbb{E}_{I \sim \mu_I}[(I^T \Phi(f, x))^2]}$$

Intuitively, given a perturbation, e.g., a perturbation on important pixels, the product (the former term) should be relatively large if the explanation indicates the important pixels too, compared to a perturbation on irrelavant pixels; while the difference (the latter term) should also be large because the model depends on important pixels to make decisions. Like this, large values would be offset by large values if the explanation is faithful to the model. Otherwise, for uniform explanations (all being constant), the former term would be a constant value and the infidelity would become large.

More details about the measure can be found in the original paper: https://arxiv.org/abs/1901.09392.

Parameters

- **model** (*callable*) – _description_
- **device** (*_type_, optional*) – _description_. Defaults to 'gpu:0'.

_build_predict_fn(*rebuild: bool = False*)

Different from InterpreterEvaluator._build_predict_fn(): using logits.

Parameters **rebuild** (*bool, optional*) – _description_. Defaults to False.

Returns _description_

Return type _type_

evaluate(*img_path: str*, *explanation: numpy.ndarray*, *recompute: bool = False*, *batch_size: int = 50*, *resize_to: int = 224*, *crop_to: int = None*)

Given `img_path`, Infidelity first generates perturbed samples, with a square removal strategy on the original image. Since the difference (the second term in the infidelity formula) is independ of the explanation, so we temporaily save these results in case this image has other explanations for evaluations.

Then, given `explanation`, we follow the formula to compute the infidelity. A normalization is added, which is not in the paper but in the official implementation.

Parameters

- **img_path** (*strornp.ndarray*) – a string for image path.
- **explanation** (*np.ndarray*) – the explanation result from an interpretation algorithm.
- **recompute** (*bool, optional*) – whether forcing to recompute. Defaults to False.
- **batch_size** (*int, optional*) – batch size for each pass.. Defaults to 50.
- **resize_to** (*int, optional*) – Images will be rescaled with the shorter edge being `resize_to`. Defaults to `224`.
- **crop_to** (*int, optional*) – After resize, images will be center cropped to a square image with the size `crop_to`. If None, no crop will be performed. Defaults to `None`.

Returns the infidelity score.

Return type int

CHAPTER 3

Model Interpretability Evaluation Metrics

3.1 PointGame

class interpretdl.**PointGame**

Pointing Game Evaluation Method.

This evaluator assumes that the explanation result should align with the visual objects. Based on this idea, the evaluation is to compute the alignment between the bounding box or semantic segmentation with the explanations.

PointGame computes the alignment to the bounding box. PointGameSegmentation computes the alignment to the semantic segmentation.

More details can be found in the original paper: https://arxiv.org/abs/1608.00507.

Note that the bounding box of annotations is required for the evaluation. This method does not need models. For API compatibility, we implement it within the same functions as other evaluators.

evaluate(*bbox: tuple*, *exp_array: numpy.ndarray*, *threshold=0.25*) → dict

Since the explanation is actually a ranking order, PointGame computes two categories of measures. One is based on thresholding. Here, `threshold` * max(`exp_array`) is used as the threshold. Based on this, precision, recall and F1 score are computed, *w.r.t.* `bbox`. Another measure does not depend on the threshold. Here, the ROC AUC score and the Average Precision (both of them are imported from `sklearn.metrics`) are computed.

Parameters

- **bbox** (*tuple*) – A tuple of four integers: (x1, y1, x2, y2), where (x1, y1) is the coordinates of the top-left point *w.r.t.* width and height respectively; (x2, y2) is the coordinates of the bottom-right point *w.r.t.* width and height respectively;
- **exp_array** (*np.ndarray*) – the explanation result from an interpretation algorithm.
- **threshold** (*float, optional*) – threshold for computing precision, recall and F1 score. Defaults to `0.25`.

Returns A dict containing `precision`, `recall`, `f1_score` and `auc_score`, `ap_score`, where the first three depend on the threshold and the last two do not.

Return type dict

3.2 PointGameSegmentation

class interpretdl.**PointGameSegmentation**

Pointing Game Evaluation Method using Segmentation.

This evaluator assumes that the explanation result should align with the visual objects. Based on this idea, the evaluation is to compute the alignment between the bounding box or semantic segmentation with the explanations.

PointGame computes the alignment to the bounding box. PointGameSegmentation computes the alignment to the semantic segmentation.

More details can be found in the original paper: https://arxiv.org/abs/1608.00507.

Note that the semantic segmentation is required for the evaluation. This method does not need models. For API compatibility, we implement it within the same functions as other evaluators.

evaluate(*seg_gt: numpy.ndarray*, *exp_array: numpy.ndarray*, *threshold=0.25*) → dict

Since the explanation is actually a ranking order, PointGameSegmentation computes two categories of measures. One is based on thresholding. Here, `threshold` * max(`exp_array`) is used as the threshold. Based on this, precision, recall and F1 score are computed, *w.r.t.* `seg_gt`. Another measure does not depend on the threshold. Here, the ROC AUC score and the Average Precision (both of them are imported from `sklearn.metrics`) are computed.

Parameters

- **seg_gt** (*np.ndarray*) – binary values are supported only currently.
- **exp_array** (*np.ndarray*) – the explanation result from an interpretation algorithm.
- **threshold** (*float, optional*) – threshold for computing precision, recall and F1 score. Defaults to `0.25`.

Returns A dict containing `precision`, `recall`, `f1_score` and `auc_score`, `ap_score`, where the first three depend on the threshold and the last two do not.

Return type dict

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index

Symbols

B

C

D

E

F

G

I

L

N

O

P

R

S

T