CLI

Contents

Anomalib CLI.

class `anomalib.cli.AnomalibCLI(`*`args=None`*`)`

Bases: `object`

Implementation of a fully configurable CLI tool for anomalib.

The advantage of this tool is its flexibility to configure the pipeline from both the CLI and a configuration file (.yaml or .json). It is even possible to use both the CLI and a configuration file simultaneously. For more details, the reader could refer to PyTorch Lightning CLI documentation.

`save_config_kwargs` is set to `overwrite=True` so that the `SaveConfigCallback` overwrites the config if it already exists.

`add_arguments_to_parser(`*`parser`*`)`

Extend trainer's arguments to add engine arguments. :rtype: `None`

Note

Since `Engine` parameters are manually added, any change to the `Engine` class should be reflected manually.

`add_export_arguments(`*`parser`*`)`

Add export arguments to the parser.

Return type:

`None`

`add_predict_arguments(`*`parser`*`)`

Add predict arguments to the parser.

Return type:

`None`

`add_subcommands(`*`**kwargs`*`)`

Initialize base subcommands and add anomalib specific on top of it.

Return type:

`None`

`add_train_arguments(`*`parser`*`)`

Add train arguments to the parser.

Return type:

`None`

`add_trainer_arguments(`*`parser, subcommand`*`)`

Add train arguments to the parser.

Return type:

`None`

`static` `anomalib_subcommands()`

Return a dictionary of subcommands and their description.

Return type:

`dict`[`str`, `dict`[`str`, `str`]]

`before_instantiate_classes()`

Modify the configuration to properly instantiate classes and sets up tiler.

Return type:

`None`

property `export`*: Callable*

Export the model using engine's export method.

property `fit`*: Callable*

Fit the model using engine's fit method.

`init_parser(**kwargs)`

Method that instantiates the argument parser.

Return type:

`ArgumentParser`

`instantiate_classes()`

Instantiate classes depending on the subcommand.

For trainer related commands it instantiates all the model, datamodule and trainer classes. But for subcommands we do not want to instantiate any trainer specific classes such as datamodule, model, etc This is because the subcommand is responsible for instantiating and executing code based on the passed config

Return type:

`None`

`instantiate_engine()`

Instantiate the engine. :rtype: `None`

> **Note**
>
> Most of the code in this method is taken from `LightningCLI`'s `instantiate_trainer` method. Refer to that method for more details.

property `predict`*: Callable*

Predict using engine's predict method.

static `subcommands()`

Skip predict subcommand as it is added later.

Return type:

`dict`[`str`, `set`[`str`]]

property `test`*: Callable*

Test the model using engine's test method.

property `train`*: Callable*

Train the model using engine's train method.

property `validate`*: Callable*

Validate the model using engine's validate method.

Previous
Callbacks

Next
Deployment