1 lines 0 Removals

124 lines + 123 Additions

```
1
```

```
import os
import os.path as osp
from typing import Callable, List, Optional

import torch

from torch_geometric.data import (
    Data,
    InMemoryDataset,
    download_url,
    extract_tar,
    extract_zip,
)


class MalNetTiny(InMemoryDataset):
    r"""The MalNet Tiny dataset from the
    `"A Large-Scale Database for Graph Representation Lea
rning"
    <https://openreview.net/pdf?id=1xDTDk3XPW>`_ paper.
    :class:`MalNetTiny` contains 5,000 malicious and beni
gn software function
    call graphs across 5 different types. Each graph cont
ains at most 5k nodes.

    Args:
        root (str): Root directory where the dataset shou
ld be saved.
        split (str, optional): If :obj:`"train"`, loads t
he training dataset.
            If :obj:`"val"`, loads the validation datase
t.
            If :obj:`"trainval"`, loads the training and
 validation dataset.
            If :obj:`"test"`, loads the test dataset.
            If :obj:`None`, loads the entire dataset.
            (default: :obj:`None`)
        transform (callable, optional): A function/transf
orm that takes in an
            :obj:`torch_geometric.data.Data` object and r
eturns a transformed
            version. The data object will be transformed
 before every access.
            (default: :obj:`None`)
        pre_transform (callable, optional): A function/tr
ansform that takes in
            an :obj:`torch_geometric.data.Data` object an
d returns a
            transformed version. The data object will be
 transformed before
            being saved to disk. (default: :obj:`None`)
        pre_filter (callable, optional): A function that
 takes in an
            :obj:`torch_geometric.data.Data` object and r
eturns a boolean
            value, indicating whether the data object sho
uld be included in the
            final dataset. (default: :obj:`None`)
    """

    data_url = ('http://malnet.cc.gatech.edu/'
                'graph-data/malnet-graphs-tiny.tar.gz')
```

```
    split_url = 'http://malnet.cc.gatech.edu/split-info/split_info_tiny.zip'
    splits = ['train', 'val', 'test']

    def __init__(
        self,
        root: str,
        split: Optional[str] = None,
        transform: Optional[Callable] = None,
        pre_transform: Optional[Callable] = None,
        pre_filter: Optional[Callable] = None,
    ):
        if split not in {'train', 'val', 'trainval', 'test', None}:
            raise ValueError(f'Split "{split}" found, but expected either '
                             f'"train", "val", "trainval", "test" or None')
        super().__init__(root, transform, pre_transform, pre_filter)
        self.data, self.slices = torch.load(self.processed_paths[0])

        if split is not None:
            split_slices = torch.load(self.processed_paths[1])
            if split == 'train':
                self._indices = range(split_slices[0], split_slices[1])
            elif split == 'val':
                self._indices = range(split_slices[1], split_slices[2])
            elif split == 'trainval':
                self._indices = range(split_slices[0], split_slices[2])
            elif split == 'test':
                self._indices = range(split_slices[2], split_slices[3])

    @property
    def raw_file_names(self) -> List[str]:
        return ['malnet-graphs-tiny', osp.join('split_info_tiny', 'type')]

    @property
    def processed_file_names(self) -> List[str]:
        return ['data.pt', 'split_slices.pt']

    def download(self):
        path = download_url(self.data_url, self.raw_dir)
        extract_tar(path, self.raw_dir)
        os.unlink(path)

        path = download_url(self.split_url, self.raw_dir)
        extract_zip(path, self.raw_dir)
        os.unlink(path)

    def process(self):
        y_map = {}
        data_list = []
        split_slices = [0]

        for split in ['train', 'val', 'test']:
            with open(osp.join(self.raw_paths[1], f'{split}.txt'), 'r') as f:
                filenames = f.read().split('\n')[:-1]
```

```
                split_slices.append(split_slices[-1] + len(filenames))

            for filename in filenames:
                path = osp.join(self.raw_paths[0], f'{filename}.edgelist')
                malware_type = filename.split('/')[0]
                y = y_map.setdefault(malware_type, len(y_map))

                with open(path, 'r') as f:
                    edges = f.read().split('\n')[5:-1]

                edge_index = [[int(s) for s in edge.split()] for edge in edges]
                edge_index = torch.tensor(edge_index).t().contiguous()
                num_nodes = int(edge_index.max()) + 1

                data = Data(edge_index=edge_index, y=y, num_nodes=num_nodes)
                data_list.append(data)

        if self.pre_filter is not None:
            data_list = [data for data in data_list if self.pre_filter(data)]

        if self.pre_transform is not None:
            data_list = [self.pre_transform(data) for data in data_list]

        torch.save(self.collate(data_list), self.processed_paths[0])
        torch.save(split_slices, self.processed_paths[1])
```