# Creating your own datasets

Although PyTorch Geometric already contains a lot of useful datasets, you may wish to create your own dataset with self-recorded or non-publicly available data.

Implementing datasets by yourself is straightforward and you may want to take a look at the source code to find out how the various datasets are implemented. However, we give a brief introduction on what is needed to setup your own dataset.

We provide two abstract classes for datasets: `torch_geometric.data.Dataset` and `torch_geometric.data.InMemoryDataset`. `torch_geometric.data.InMemoryDataset` inherits from `torch_geometric.data.Dataset` and should be used if the whole dataset fits into memory.

Following the `torchvision` convention, each dataset gets passed a root folder which indicates where the dataset should be stored. We split up the root folder into two folders: the `raw_dir`, where the dataset gets downloaded to, and the `processed_dir`, where the processed dataset is being saved.

In addition, each dataset can be passed a `transform`, a `pre_transform` and a `pre_filter` function, which are `None` by default. The `transform` function dynamically transforms the data object before accessing (so it is best used for data augmentation). The `pre_transform` function applies the transformation before saving the data objects to disk (so it is best used for heavy precomputation which needs to be only done once). The `pre_filter` function can manually filter out data objects before saving. Use cases may involve the restriction of data objects being of a specific class.

## Creating “in memory datasets”

In order to create a `torch_geometric.data.InMemoryDataset`, you need to implement four fundamental methods:

`torch_geometric.data.InMemoryDataset.raw_file_names()`:

A list of files in the `raw_dir` which needs to be found in order to skip the download.

`torch_geometric.data.InMemoryDataset.processed_file_names()`:

A list of files in the `processed_dir` which needs to be found in order to skip the processing.

`torch_geometric.data.InMemoryDataset.download()`:

Downloads raw data into `raw_dir`.

`torch_geometric.data.InMemoryDataset.process()`:

Processes raw data and saves it into the `processed_dir`.

You can find helpful methods to download and extract data in `torch_geometric.data`.

The real magic happens in the body of `torch_geometric.data.InMemoryDataset.process()`. Here, we need to read and create a list of `torch_geometric.data.Data` objects and save it into the `processed_dir`. Because saving a huge python list is really slow, we collate the list into one huge `torch_geometric.data.Data` object via `torch_geometric.data.InMemoryDataset.collate()` before saving . The collated data object has concatenated all examples into one big data object and, in addition, returns a `slices` dictionary to reconstruct single examples from this object. Finally, we need to load these two objects in the constructor into the properties `self.data` and `self.slices`.

Let's see this process in a simplified example:

```
import torch
from torch_geometric.data import InMemoryDataset


class MyOwnDataset(InMemoryDataset):
    def __init__(self, root, transform=None, pre_transform=None):
        super(MyOwnDataset, self).__init__(root, transform, pre_transform)
        self.data, self.slices = torch.load(self.processed_paths[0])

    @property
    def raw_file_names(self):
        return ['some_file_1', 'some_file_2', ...]

    @property
    def processed_file_names(self):
        return ['data.pt']

    def download(self):
        # Download to `self.raw_dir`.

    def process(self):
        # Read data into huge `Data` list.
        data_list = [...]

        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]

        data, slices = self.collate(data_list)
        torch.save((data, slices), self.processed_paths[0])
```

## Creating "larger" datasets

For creating datasets which do not fit into memory, the `torch_geometric.data.Dataset` must be used, where we closely follow the concepts of the `torchvision` datasets.

Therefore, the following methods need to be further implemented:

`torch_geometric.data.Dataset.__len__()`:

Returns the number of examples in your dataset.

`torch_geometric.data.Dataset.get()`:

Implements the logic to load a single graph.

Internally, `torch_geometric.data.Dataset.__getitem__()` gets data objects from `torch_geometric.data.Dataset.get()` and optionally transforms them according to `transform`.

Let's see this process in a simplified example:

```
import os.path as osp

import torch
from torch_geometric.data import Dataset


class MyOwnDataset(Dataset):
    def __init__(self, root, transform=None, pre_transform=None):
        super(MyOwnDataset, self).__init__(root, transform, pre_transform)

    @property
    def raw_file_names(self):
        return ['some_file_1', 'some_file_2', ...]

    @property
    def processed_file_names(self):
        return ['data_1.pt', 'data_2.pt', ...]

    def __len__(self):
        return len(self.processed_file_names)

    def download(self):
        # Download to `self.raw_dir`.

    def process(self):
        i = 0
        for raw_path in self.raw_paths:
            # Read data from `raw_path`.
            data = Data(...)

            if self.pre_filter is not None and not self.pre_filter(data):
                continue

            if self.pre_transform is not None:
                data = self.pre_transform(data)

            torch.save(data, ops.join(self.processed_dir, 'data_{}.pt'.format(i)))
            i += 1

    def get(self, idx):
        data = torch.load(osp.join(self.processed_dir, 'data_{}.pt'.format(idx))
        return data
```

Here, each graph data object gets saved individually in `torch_geometric.data.Dataset.process()`, and is manually loaded in `torch_geometric.data.Dataset.get()`.

# Frequently Asked Questions

1. **How can I skip the execution of `download()` and/or `process()`?**

   You can skip downloading and/or processing by overriding the `_download()` and `_process()` methods:

   ```
   class MyOwnDataset(Dataset):
       def __init__(self, root, transform=None, pre_transform=None):
           super(MyOwnDataset, self).__init__(root, transform, pre_transform)

   def _download(self):
       pass

   def _process(self):
       pass
   ```

2. **Do I really need to use these dataset interfaces?**

   No! Just as in regular PyTorch, you do not have to use datasets, *e.g.*, when you want to create synthetic data on the fly without saving them explicitly to disk. In this case, simply pass a regular python list holding `torch_geometric.data.Data` objects and pass them to `torch_geometric.data.DataLoader`:

   ```
   from torch_geometric.data import Data, DataLoader

   data_list = [Data(...), ..., Data(...)]
   loader = DataLoader(data_list, batch_size=32)
   ```