Contents

Transforms are a general way to modify and customize `Data` or `HeteroData` objects, either by implicitly passing them as an argument to a `Dataset`, or by applying them explicitly to individual `Data` or `HeteroData` objects:

```
import torch_geometric.transforms as T
from torch_geometric.datasets import TUDataset

transform = T.Compose([T.ToUndirected(), T.AddSelfLoops()])

dataset = TUDataset(path, name='MUTAG', transform=transform)
data = dataset[0]  # Implicitly transform data on every access.

data = TUDataset(path, name='MUTAG')[0]
data = transform(data)  # Explicitly transform data.
```

General Transforms

`BaseTransform`	An abstract base class for writing transforms.
`Compose`	Composes several transforms together.
`ComposeFilters`	Composes several filters together.
`ToDevice`	Performs tensor device conversion, either for all attributes of the `Data` object or only the ones given by `attrs` (functional name: `to_device`).

`ToSparseTensor`	Converts the `edge_index` attributes of a homogeneous or heterogeneous data object into a **transposed** `torch_sparse.SparseTensor` or PyTorch `torch.sparse.Tensor` object with key `adj_t` (functional name: `to_sparse_tensor`).
`Constant`	Appends a constant value to each node feature `x` (functional name: `constant`).
`NormalizeFeatures`	Row-normalizes the attributes given in `attrs` to sum-up to one (functional name: `normalize_features`).
`SVDFeatureReduction`	Dimensionality reduction of node features via Singular Value Decomposition (SVD) (functional name: `svd_feature_reduction`).
`RemoveTrainingClasses`	Removes classes from the node-level training set as given by `data.train_mask`, *e.g.*, in order to get a zero-shot label scenario (functional name: `remove_training_classes`).
`RandomNodeSplit`	Performs a node-level random split by adding `train_mask`, `val_mask` and `test_mask` attributes to the `Data` or `HeteroData` object (functional name: `random_node_split`).
`RandomLinkSplit`	Performs an edge-level random split into training, validation and test sets of a `Data` or a `HeteroData` object (functional name: `random_link_split`).
`NodePropertySplit`	Creates a node-level split with distributional shift based on a given node property, as proposed in the "Evaluating Robustness and Uncertainty of Graph Models Under Structural Distributional Shifts" paper (functional name: `node_property_split`).
`IndexToMask`	Converts indices to a mask representation (functional name: `index_to_mask`).
`MaskToIndex`	Converts a mask to an index representation (functional name: `mask_to_index`).
`Pad`	Applies padding to enforce consistent tensor shapes (functional name: `pad`).

Graph Transforms

`ToUndirected`	Converts a homogeneous or heterogeneous graph to an undirected graph such that $(j, i) \in \mathcal{E}$ for every edge $(i, j) \in \mathcal{E}$ (functional name: `to_undirected`).
`OneHotDegree`	Adds the node degree as one hot encodings to the node features (functional name: `one_hot_degree`).
`TargetIndegree`	Saves the globally normalized degree of target nodes (fu `target_indegree`).

`LocalDegreeProfile`	Appends the Local Degree Profile (LDP) from the "A Simple yet Effective Baseline for Non-attribute Graph Classification" paper (functional name: `local_degree_profile`).
`AddSelfLoops`	Adds self-loops to the given homogeneous or heterogeneous graph (functional name: `add_self_loops`).
`AddRemainingSelfLoops`	Adds remaining self-loops to the given homogeneous or heterogeneous graph (functional name: `add_remaining_self_loops`).
`RemoveSelfLoops`	Removes all self-loops in the given homogeneous or heterogeneous graph (functional name: `remove_self_loops`).
`RemoveIsolatedNodes`	Removes isolated nodes from the graph (functional name: `remove_isolated_nodes`).
`RemoveDuplicatedEdges`	Removes duplicated edges from a given homogeneous or heterogeneous graph.
`KNNGraph`	Creates a k-NN graph based on node positions `data.pos` (functional name: `knn_graph`).
`RadiusGraph`	Creates edges based on node positions `data.pos` to all points within a given distance (functional name: `radius_graph`).
`ToDense`	Converts a sparse adjacency matrix to a dense adjacency matrix with shape `[num_nodes, num_nodes, *]` (functional name: `to_dense`).
`TwoHop`	Adds the two hop edges to the edge indices (functional name: `two_hop`).
`LineGraph`	Converts a graph to its corresponding line-graph (functional name: `line_graph`).
`LaplacianLambdaMax`	Computes the highest eigenvalue of the graph Laplacian given by `torch_geometric.utils.get_laplacian()` (functional name: `laplacian_lambda_max`).
`GDC`	Processes the graph via Graph Diffusion Convolution (GDC) from the "Diffusion Improves Graph Learning" paper (functional name: `gdc`).
`SIGN`	The Scalable Inception Graph Neural Network module (SIGN) from the "SIGN: Scalable Inception Graph Neural Networks" paper (functional name: `sign`), which precomputes the fixed representations.
`GCNNorm`	Applies the GCN normalization from the "Semi-supervised Classification with Graph Convolutional Networks" paper (functional name: `gcn_norm`).

`AddMetaPaths`	Adds additional edge types to a `HeteroData` object between the source node type and the destination node type of a given `metapath`, as described in the "Heterogenous Graph Attention Networks" paper (functional name: `add_metapaths`).
`AddRandomMetaPaths`	Adds additional edge types similar to `AddMetaPaths`.
`RootedEgoNets`	Collects rooted k-hop EgoNets for each node in the graph, as described in the "From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness" paper.
`RootedRWSubgraph`	Collects rooted random-walk based subgraphs for each node in the graph, as described in the "From Stars to Subgraphs: Uplifting Any GNN with Local Structure Awareness" paper.
`LargestConnectedComponents`	Selects the subgraph that corresponds to the largest connected components in the graph (functional name: `largest_connected_components`).
`VirtualNode`	Appends a virtual node to the given homogeneous graph that is connected to all other nodes, as described in the "Neural Message Passing for Quantum Chemistry" paper (functional name: `virtual_node`).
`AddLaplacianEigenvectorPE`	Adds the Laplacian eigenvector positional encoding from the "Benchmarking Graph Neural Networks" paper to the given graph (functional name: `add_laplacian_eigenvector_pe`).
`AddRandomWalkPE`	Adds the random walk positional encoding from the "Graph Neural Networks with Learnable Structural and Positional Representations" paper to the given graph (functional name: `add_random_walk_pe`).
`FeaturePropagation`	The feature propagation operator from the "On the Unreasonable Effectiveness of Feature propagation in Learning on Graphs with Missing Node Features" paper (functional name: `feature_propagation`).
`HalfHop`	The graph upsampling augmentation from the "Half-Hop: A Graph Upsampling Approach for Slowing Down Message Passing" paper.

Vision Transforms

`Distance`	Saves the Euclidean distance of linked nodes in its edge attributes (functional name: `distance`).
`Cartesian`	Saves the relative Cartesian coordinates of linked nodes in its edge attributes (functional name: `cartesian`).
`LocalCartesian`	Saves the relative Cartesian coordinates of linked nodes in its (functional name: `local_cartesian`).

`Polar`	Saves the polar coordinates of linked nodes in its edge attributes (functional name: `polar`).
`Spherical`	Saves the spherical coordinates of linked nodes in its edge attributes (functional name: `spherical`).
`PointPairFeatures`	Computes the rotation-invariant Point Pair Features (functional name: `point_pair_features`).
`Center`	Centers node positions `data.pos` around the origin (functional name: `center`).
`NormalizeRotation`	Rotates all points according to the eigenvectors of the point cloud (functional name: `normalize_rotation`).
`NormalizeScale`	Centers and normalizes node positions to the interval $(-1, 1)$ (functional name: `normalize_scale`).
`RandomJitter`	Translates node positions by randomly sampled translation values within a given interval (functional name: `random_jitter`).
`RandomFlip`	Flips node positions along a given axis randomly with a given probability (functional name: `random_flip`).
`LinearTransformation`	Transforms node positions `data.pos` with a square transformation matrix computed offline (functional name: `linear_transformation`).
`RandomScale`	Scales node positions by a randomly sampled factor s within a given interval, *e.g.*, resulting in the transformation matrix (functional name: `random_scale`).
`RandomRotate`	Rotates node positions around a specific axis by a randomly sampled factor within a given interval (functional name: `random_rotate`).
`RandomShear`	Shears node positions by randomly sampled factors s within a given interval, *e.g.*, resulting in the transformation matrix (functional name: `random_shear`).
`FaceToEdge`	Converts mesh faces of shape `[3, num_faces]` or `[4, num_faces]` to edge indices of shape `[2, num_edges]` (functional name: `face_to_edge`).
`SamplePoints`	Uniformly samples a fixed number of points on the mesh faces according to their face area (functional name: `sample_points`).
`FixedPoints`	Samples a fixed number of points and features from a point cloud (functional name: `fixed_points`).
`GenerateMeshNormals`	Generate normal vectors for each mesh node based on neighboring faces (functional name: `generate_mesh_normals`).

`Delaunay`	Computes the delaunay triangulation of a set of points (functional name: `delaunay`).
`ToSLIC`	Converts an image to a superpixel representation using the `skimage.segmentation.slic()` algorithm, resulting in a `torch_geometric.data.Data` object holding the centroids of superpixels in `data.pos` and their mean color in `data.x` (functional name: `to_slic`).
`GridSampling`	Clusters points into fixed-sized voxels (functional name: `grid_sampling`).