torch_geometric.transforms

class `Compose`(*transforms*) [source]

Composes several transforms together.

Parameters: **transforms** (list of `transform` objects) – List of transforms to compose.

class `Constant`(*value=1*, *cat=True*) [source]

Adds a constant value to each node feature.

Parameters:

- **value** (*int*, *optional*) – The value to add. (default: `1`)
- **cat** (*bool*, *optional*) – If set to `False`, all existing node features will be replaced. (default: `True`)

class `Distance`(*norm=True*, *max_value=None*, *cat=True*) [source]

Saves the Euclidean distance of linked nodes in its edge attributes.

Parameters:

- **norm** (*bool*, *optional*) – If set to `False`, the output will not be normalized to the interval $[0, 1]$. (default: `True`)
- **max_value** (*float*, *optional*) – If set and `norm=True`, normalization will be performed based on this value instead of the maximum value found in the data. (default: `None`)
- **cat** (*bool*, *optional*) – If set to `False`, all existing edge attributes will be replaced. (default: `True`)

class `Cartesian`(*norm=True*, *max_value=None*, *cat=True*) [source]

Saves the relative Cartesian coordinates of linked nodes in its edge attributes.

Parameters:

- **norm** (*bool*, *optional*) – If set to `False`, the output will not be normalized to the interval $[0, 1]^D$. (default: `True`)
- **max_value** (*float*, *optional*) – If set and `norm=True`, normalization will be performed based on this value instead of the maximum value found in the data. (default: `None`)
- **cat** (*bool*, *optional*) – If set to `False`, all existing edge attributes will be replaced. (default: `True`)

class **LocalCartesian**(*cat=True*) [source]

Saves the relative Cartesian coordinates of linked nodes in its edge attributes. Each coordinate gets *neighborhood-normalized* to the interval $[0, 1]^D$.

Parameters: **cat** (*bool, optional*) – If set to `False`, all existing edge attributes will be replaced. (default: `True`)

class **Polar**(*norm=True, max_value=None, cat=True*) [source]

Saves the relative polar coordinates of linked nodes in its edge attributes.

Parameters:

- **norm** (*bool, optional*) – If set to `False`, the output will not be normalized to the interval $[0, 1]^2$. (default: `True`)
- **max_value** (*float, optional*) – If set and `norm=True`, normalization will be performed based on this value instead of the maximum value found in the data. (default: `None`)
- **cat** (*bool, optional*) – If set to `False`, all existing edge attributes will be replaced. (default: `True`)

class **Spherical**(*norm=True, max_value=None, cat=True*) [source]

Saves the relative spherical coordinates of linked nodes in its edge attributes.

Parameters:

- **norm** (*bool, optional*) – If set to `False`, the output will not be normalized to the interval $[0, 1]^3$. (default: `True`)
- **max_value** (*float, optional*) – If set and `norm=True`, normalization will be performed based on this value instead of the maximum value found in the data. (default: `None`)
- **cat** (*bool, optional*) – If set to `False`, all existing edge attributes will be replaced. (default: `True`)

class **PointPairFeatures**(*cat=True*) [source]

Computes the rotation-invariant Point Pair Features

$$\left(\|\mathbf{d_{j,i}}\|, \angle(\mathbf{n}_i, \mathbf{d_{j,i}}), \angle(\mathbf{n}_j, \mathbf{d_{j,i}}), \angle(\mathbf{n}_i, \mathbf{n}_j)\right)$$

of linked nodes in its edge attributes, where $\mathbf{d}_{j,i}$ denotes the difference vector between, and $\mathbf{n}_i$ and $\mathbf{n}_j$ denote the surface normals of node i and j respectively.

Parameters: **cat** (*bool, optional*) – If set to `False`, all existing edge attributes will be replaced. (default: `True`) *class* **OneHotDegree**(*max_degree*, *cat=True*) [source]

Adds the node degree as one hot encodings to the node features.

Parameters:
- **max_degree** (*int*) – Maximum degree.
- **cat** (*bool, optional*) – Concat node degrees to node features instead of replacing them. (default: `True`)

class **TargetIndegree**(*norm=True*, *max_value=None*, *cat=True*) [source]

Saves the globally normalized degree of target nodes

$$\mathbf{u}(i,j) = \frac{\deg(j)}{\max_{v \in \mathcal{V}} \deg(v)}$$

in its edge attributes.

Parameters: **cat** (*bool, optional*) – Concat pseudo-coordinates to edge attributes instead of replacing them. (default: `True`)

class **LocalDegreeProfile** [source]

Appends the Local Degree Profile (LDP) from the “A Simple yet Effective Baseline for Non-attribute Graph Classification” paper

$$\mathbf{x}_i = \mathbf{x}_i \, \Vert \, (\deg(i), \min(DN(i)), \max(DN(i)), \textrm{mean}(DN(i)), \textrm{std}(DN(i)))$$

to the node features, where $DN(i) = \{ \deg(j) \mid j \in \mathcal{N}(i) \}$.

class **Center** [source]

Centers node positions around the origin.

class **NormalizeRotation**(*max_points=-1*) [source]

Rotates all points so that the eigenvectors overlie the axes of the Cartesian coordinate system. If the data additionally holds normals saved in `data.norm` these will be also rotated.

Parameters: **max_points** (*int, optional*) – If set to a value greater than `0`, only a random number of `max_points` points are sampled and used to compute eigenvectors. (default: `-1`)

class **NormalizeScale** [source]

Centers and normalizes node positions to the interval $(-1, 1)$.

class **RandomTranslate**(*translate*) [source]

Translates node positions by randomly sampled translation values within a given interval. In contrast to other random transformations, translation is applied separately at each position.

Parameters: **translate** (*sequence or float or int*) – Maximum translation in each dimension, defining the range $(-\mathrm{translate}, +\mathrm{translate})$ to sample from. If `translate` is a number instead of a sequence, the same range is used for each dimension.

class **RandomFlip**(*axis*, *p=0.5*) [source]

Flips node positions along a given axis randomly with a given probability.

Parameters:

- **axis** (*int*) – The axis along the position of nodes being flipped.
- **p** (*float, optional*) – Probability that node positions will be flipped. (default: `0.5`)

class **LinearTransformation**(*matrix*) [source]

Transforms node positions with a square transformation matrix computed offline.

Parameters: **matrix** (*Tensor*) – tensor with shape $[D, D]$ where D corresponds to the dimensionality of node positions.

class **RandomScale**(*scales*) [source]

Scales node positions by a randomly sampled factor s within a given interval, *e.g.*, resulting in the transformation matrix

$$\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & s \end{bmatrix}$$

for three-dimensional positions.

Parameters: **scale** (*tuple*) – scaling factor interval, e.g. `(a, b)`, then scale is randomly sampled from the range $a \leq \mathrm{scale} \leq b$.

class **RandomRotate**(*degrees*, *axis=0*) [source]

Rotates node positions around a specific axis by a randomly sampled factor within a given interval.

Loading [MathJax]/jax/output/HTML-CSS/jax.js

Parameters:
- **degrees** (*tuple or float*) – Rotation interval from which the rotation angle is sampled. If `degrees` is a number instead of a tuple, the interval is given by $[-\mathrm{degrees}, \mathrm{degrees}]$.
- **axis** (*int, optional*) – The rotation axis. (default: `0`)

class **`RandomShear`**(*shear*) [source]

Shears node positions by randomly sampled factors s within a given interval, *e.g.*, resulting in the transformation matrix

$$\begin{bmatrix} 1 & s_{xy} & s_{xz} \\ s_{yx} & 1 & s_{yz} \\ s_{zx} & z_{zy} & 1 \end{bmatrix}$$

for three-dimensional positions.

Parameters: **shear** (*float or int*) – maximum shearing factor defining the range $(-\mathrm{shear}, +\mathrm{shear})$ to sample from.

class **`NormalizeFeatures`** [source]

Row-normalizes node features to sum-up to one.

class **`AddSelfLoops`** [source]

Adds self-loops to edge indices.

class **`KNNGraph`**(*k=6, loop=False, force_undirected=False*) [source]

Creates a k-NN graph based on node positions.

Parameters:
- **k** (*int, optional*) – The number of neighbors. (default: `6`)
- **loop** (*bool, optional*) – If `True`, the graph will contain self-loops. (default: `False`)
- **force_undirected** (*bool, optional*) – If set to `True`, new edges will be undirected. (default: `False`)

class **`RadiusGraph`**(*r, loop=False, max_num_neighbors=32*) [source]

Creates edges based on node positions to all points within a given distance.

Parameters:

- **r** (*float*) – The distance.
- **loop** (*bool, optional*) – If `True`, the graph will contain self-loops. (default: `False`)
- **max_num_neighbors** (*int, optional*) – The maximum number of neighbors to return for each element in `y`. This flag is only needed for CUDA tensors. (default: `32`)

class **FaceToEdge**(*remove_faces=True*) [source]

Converts mesh faces `[3, num_faces]` to edge indices `[2, num_edges]`.

Parameters: **remove_faces** (*bool, optional*) – If set to `False`, the face tensor will not be removed.

class **SamplePoints**(*num, remove_faces=True, include_normals=False*) [source]

Uniformly samples `num` points on the mesh faces according to their face area.

Parameters:

- **num** (*int*) – The number of points to sample.
- **remove_faces** (*bool, optional*) – If set to `False`, the face tensor will not be removed. (default: `True`)
- **include_normals** (*bool, optional*) – If set to `True`, then compute normals for each sampled point. (default: `False`)

class **ToDense**(*num_nodes=None*) [source]

Converts a sparse adjacency matrix to a dense adjacency matrix with shape `[num_nodes, num_nodes, *]`.

Parameters: **num_nodes** (*int*) – The number of nodes. If set to `None`, the number of nodes will get automatically inferred. (default: `None`)

class **TwoHop** [source]

Adds the two hop edges to the edge indices.

class **LineGraph**(*force_directed=False*) [source]

Converts a graph to its corresponding line-graph:

$$\begin{aligned} L(\mathrm{G}) &= (\mathrm{V}', \mathrm{E}') \\ \mathrm{V}' &= \mathrm{E} \\ \mathrm{E}' &= \{(e_1, e_2) : e_1 \cap e_2 \neq \emptyset\} \end{aligned}$$

Line-graph node indices are equal to indices in the original graph's coalesced `edge_index`. For undirected graphs, the maximum line-graph node index is `(data.edge_index.size(1) // 2) - 1`.

New node features are given by old edge attributes. For undirected graphs, edge attributes for reciprocal edges `(row, col)` and `(col, row)` get summed together.

Parameters: **force_directed** (*bool, optional*) – If set to `True`, the graph will be always treated as a directed graph. (default: `False`)

class **GenerateMeshNormals** [source]

Generate normal vectors for each mesh node based on neighboring faces.