Double-click (or enter) to edit

Please choose a graph-based GNN model of your choice to classify (quark/gluon) jets. Proceed as follows:

- Convert the images into a point cloud dataset by only considering the non-zero pixels for every event.
- Cast the point cloud data into a graph representation by coming up with suitable representations for nodes and edges.
- Train your model on the obtained graph representations of the jet events.

Discuss the resulting performance of the chosen architecture.

## Loading Quark-Gluon Data :

Loading the hdf5 file using `h5py`'s `File` object with the Standard Driver appropriate for the current platform.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
Start coding or generate with AI.
```

```
import numpy as np
import h5py

def load_h5(file_name, size):
    # Load the dataset from the HDF5 file
    with h5py.File(file_name, 'r') as f:
        X = np.array(f['X_jets'][:size])
        y = np.array(f['y'][:size])
    return X, y
```

```
Start coding or generate with AI.
```

```
file = '/content/drive/MyDrive/quark-gluon_data-set_n139306.hdf5'
size = 16000
X, y = load_h5(file, size)
```

```
# Count occurrences of each value (0s and 1s)
y_int = y.astype(np.int64)
counts = np.bincount(y_int)

# Create a dictionary to store counts
counts_dict = {'0': counts[0], '1': counts[1]}

print(counts_dict)
del y_int
del counts
del counts_dict
```

```
{'0': 7953, '1': 8047}
```

```
!pip install torch-geometric
```

```
Collecting torch-geometric
  Downloading torch_geometric-2.5.2-py3-none-any.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 11.6 MB/s eta 0:00:00
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (4.66.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (1.25.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (1.11.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (2023.6.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (3.1.3)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (3.9.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (2.31.0)
Requirement already satisfied: pyparsing in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (3.1.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (1.2.2)
Requirement already satisfied: psutil>=5.8.0 in /usr/local/lib/python3.10/dist-packages (from torch-geometric) (5.9.5)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (6.0.5
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->torch-geometric) (4
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch-geometric) (2.1.5)
```

```
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric) (2.0.7
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torch-geometric) (2024
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->torch-geometric) (1.3.2
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn->torch-geometric)
Installing collected packages: torch-geometric
Successfully installed torch-geometric-2.5.2
```

Importing necessary libaries and modules as required for the immediate tasks ->

```
!pip install pytorch-lightning
```

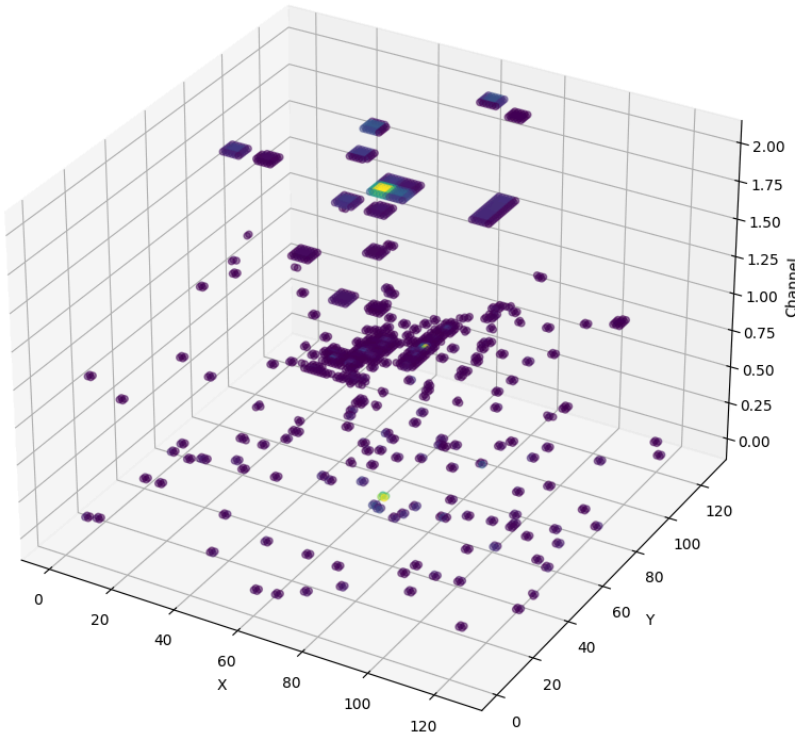
```
Collecting pytorch-lightning
  Downloading pytorch_lightning-2.2.1-py3-none-any.whl (801 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 801.6/801.6 kB 9.7 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17.2 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (1.25.2)
Requirement already satisfied: torch>=1.13.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (2.2.1+cu121)
Requirement already satisfied: tqdm>=4.57.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (4.66.2)
Requirement already satisfied: PyYAML>=5.4 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (6.0.1)
Requirement already satisfied: fsspec[http]>=2022.5.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (2023.6
Collecting torchmetrics>=0.7.0 (from pytorch-lightning)
  Downloading torchmetrics-1.3.2-py3-none-any.whl (841 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 841.5/841.5 kB 15.3 MB/s eta 0:00:00
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (24.0)
Requirement already satisfied: typing-extensions>=4.4.0 in /usr/local/lib/python3.10/dist-packages (from pytorch-lightning) (4.10
Collecting lightning-utilities>=0.8.0 (from pytorch-lightning)
  Downloading lightning_utilities-0.11.0-py3-none-any.whl (25 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from fsspec[http]>=2022.5.0->pytorch-lightnin
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]>=2022.5.0
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->pytorch-li
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->pytorch-lightning) (3.13.
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->pytorch-lightning) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->pytorch-lightning) (3.2.1
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->pytorch-lightning) (3.1.3)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (23.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 23.7/23.7 MB 55.3 MB/s eta 0:00:00
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (823 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 823.6/823.6 kB 67.3 MB/s eta 0:00:00
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (14.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.1/14.1 MB 81.7 MB/s eta 0:00:00
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (731.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 731.7/731.7 MB 1.8 MB/s eta 0:00:00
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (410.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.6/410.6 MB 2.9 MB/s eta 0:00:00
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (121.6 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 121.6/121.6 MB 14.2 MB/s eta 0:00:00
Collecting nvidia-curand-cu12==10.3.2.106 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl (56.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.5/56.5 MB 29.4 MB/s eta 0:00:00
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.whl (124.2 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 124.2/124.2 MB 13.8 MB/s eta 0:00:00
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.whl (196.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 196.0/196.0 MB 5.5 MB/s eta 0:00:00
Collecting nvidia-nccl-cu12==2.19.3 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_nccl_cu12-2.19.3-py3-none-manylinux1_x86_64.whl (166.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 166.0/166.0 MB 10.3 MB/s eta 0:00:00
Collecting nvidia-nvtx-cu12==12.1.105 (from torch>=1.13.0->pytorch-lightning)
  Downloading nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.1/99.1 kB 15.6 MB/s eta 0:00:00
Requirement already satisfied: triton==2.2.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->pytorch-lightning) (
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->torch>=1.13.0->pytorch-lightning)
```

```
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from torch_geometric.data import Data, Batch
from torch_geometric.loader import DataLoader
from torch.optim import Adam
import torch.optim as optim
import pytorch_lightning as pl
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch_geometric.nn import SAGEConv
from torch_geometric.nn import global_mean_pool
from torch.nn import Linear
```

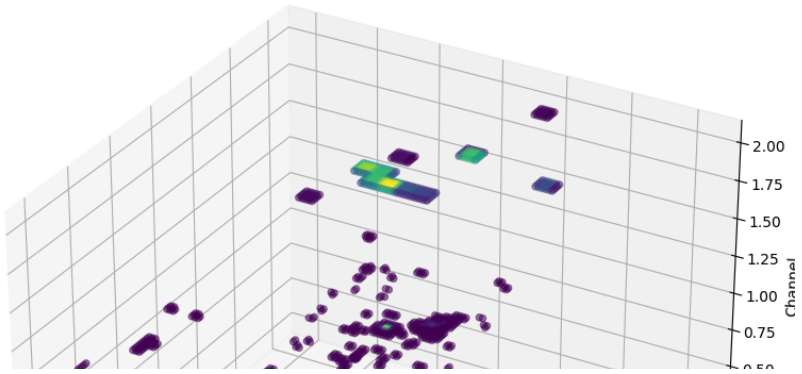
```
from skimage.transform import resize
from sklearn.preprocessing import normalize

def data_preprocess(X_jets):
    #Normalizing the images

    # Resizing images from (125, 125, 3) to (128, 128, 3)
    resized_images = np.zeros((X_jets.shape[0], 128, 128, 3), dtype=np.float32)
    for i in range(X_jets.shape[0]):
        resized_images[i] = resize(X_jets[i], (128, 128), anti_aliasing=True)

    X_jets = resized_images
    del resized_images

    # Normalizing the entire image across all channels
    mean = np.mean(X_jets)
    std = np.std(X_jets)
    X_jets = (X_jets - mean) / std

    # Assuming X_jets is your image array
    X_jets = np.clip(X_jets, 0, None)
    return X_jets


X = data_preprocess(X)


#combined.shape
```

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def plot_point_clouds(X, y):
    # Store the point clouds for all images in a list
    point_clouds = []

    for i in range(5): # Limit to the first 5 images
        # Extract the non-zero pixel coordinates and values for each channel
        non_zero_Tracks = np.nonzero(X[i, :, :, 0])
        non_zero_ECAL = np.nonzero(X[i, :, :, 1])
        non_zero_HCAL = np.nonzero(X[i, :, :, 2])

        # Create a 2D array, where each row represents the coordinate of the non-zero entity.
        coords_Tracks = np.column_stack(non_zero_Tracks)
        coords_ECAL = np.column_stack(non_zero_ECAL)
        coords_HCAL = np.column_stack(non_zero_HCAL)

        # For visualization placing Tracks, ECAL, and HCAL on z = 0,1,2 respectively.
        values_Tracks = X[i, non_zero_Tracks[0], non_zero_Tracks[1], 0]
        values_ECAL = X[i, non_zero_ECAL[0], non_zero_ECAL[1], 1]
        values_HCAL = X[i, non_zero_HCAL[0], non_zero_HCAL[1], 2]

        coords_Tracks = np.hstack((coords_Tracks, np.zeros((coords_Tracks.shape[0], 1))))
        coords_ECAL = np.hstack((coords_ECAL, np.zeros((coords_ECAL.shape[0], 1))))
        coords_HCAL = np.hstack((coords_HCAL, np.zeros((coords_HCAL.shape[0], 1))))

        # Store the point cloud for this image in the list
        point_clouds.append({'tracks': (coords_Tracks, values_Tracks), 'ECAL': (coords_ECAL, values_ECAL), 'HCAL': (coords_HCAL, values_

        # Plot the point cloud for this image
        fig = plt.figure(figsize=(10, 10))
        ax = fig.add_subplot(111, projection='3d')

        ax.scatter(coords_Tracks[:, 0], coords_Tracks[:, 1], coords_Tracks[:, 2], c=values_Tracks, cmap='viridis', alpha=0.5)
        ax.scatter(coords_ECAL[:, 0], coords_ECAL[:, 1], coords_ECAL[:, 2] + 1, c=values_ECAL, cmap='viridis', alpha=0.5)
        ax.scatter(coords_HCAL[:, 0], coords_HCAL[:, 1], coords_HCAL[:, 2] + 2, c=values_HCAL, cmap='viridis', alpha=0.5)

        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Channel')
        ax.set_title('Point Cloud Visualization for Image {}'.format(i))

        plt.show()

#
# del non_zero_Tracks
# del non_zero_ECAL
# del non_zero_HCAL
# del coords_Tracks
# del coords_ECAL
# del coords_HCAL
# del values_Tracks
# del values_ECAL
# del values_HCAL
# #del point_clouds
# del point_cloud


plot_point_clouds(X, y)
```

Point Cloud Visualization for Image 0




Point Cloud Visualization for Image 1



Now, the question comes -> how to suitably cast this point cloud data as a suitable graph data structure.

- We need a `node` construct, and an `edge` construct to atleast start off.
- Either make each pixel a `node`, or construct some cluster to acquire a node.
- Initially, we make each pixel a node, and invoke the `kneighbors_graph` function to construct the edges around these nodes, joining `n=2` nearest neighbors with each node.

`dataset` entity contains a graph for each of the loaded images, which we further split and load in train, test and validation loaders.

Instead of working on individual points like PointNet, exploit local geometric structures by constructing a local neighborhood graph and applying convolution-like operations on the edges connecting neighboring pairs of points, in the spirit of GNNs.

This gives us the added advantage of exploiting the nonlocal information that diffuses through the point cloud's updated feature spaces in subsequent layers.

```
def return_nonzero(X):
    # Reshape the data to be compatible with torch_geometric
    orig_len = X.shape[1]
    orig_wid = X.shape[2]
    data = X.reshape((-1, orig_len* orig_wid, 3))

    #Making a bool array to find which indices have non-zero values
    non_black_pixels_mask = np.any(data != [0., 0., 0.], axis=-1)

    mask_reshaped = non_black_pixels_mask.reshape((-1, orig_len, orig_wid))
    return mask_reshaped
```

```
mask_reshaped = return_nonzero(X)


def create_graph_feats(mask_reshaped):
    indices_list = []
    features_list = []
    for i, x in enumerate(mask_reshaped):
        true_indices = np.where(x)
        feature_vec = X[i, true_indices[0], true_indices[1], :]
        indices_array = np.column_stack((true_indices[0], true_indices[1]))
        indices_list.append(indices_array)
        features_list.append(feature_vec)
    return indices_list, features_list



indices_list, features_list = create_graph_feats(mask_reshaped)


print(features_list[0].shape)
```

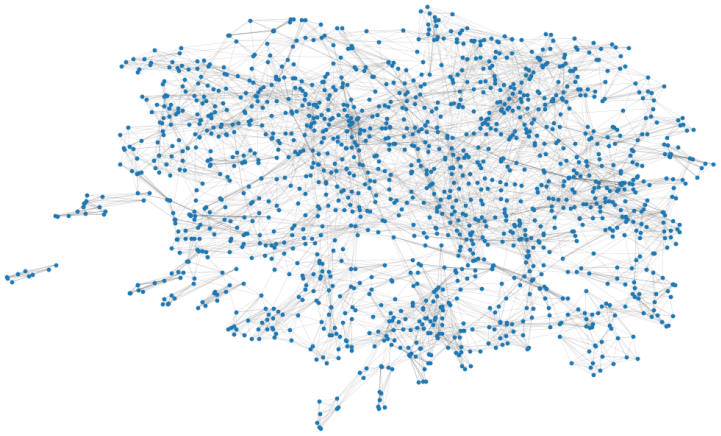
```
(1529, 3)
```

```
import argparse
import numpy as np
import scipy.sparse as sp


def import_class(name):
    try:
        components = name.split('.')
        module = __import__(components[0])
        for c in components[1:]:
            module = getattr(module, c)
    except AttributeError:
        module = None
    return module

def str2bool(v):
    if v.lower() in ('yes', 'true', 't', 'y', '1'):
        return True
    elif v.lower() in ('no', 'false', 'f', 'n', '0'):
        return False
    else:
        raise argparse.ArgumentTypeError('Boolean value expected.')


def get_total_parameters(model):
    total = sum(p.numel() for p in model.parameters())
    trainable = sum(p.numel() for p in model.parameters() if p.requires_grad)
    return {'Total': total, 'Trainable': trainable}


def _get_weights(dist, indices):
    num_nodes, k = dist.shape
    assert num_nodes, k == indices.shape
    assert dist.min() >= 0
    # weight matrix
    sigma2 = np.mean(dist[:, -1])**2
    dist = np.exp(-dist**2 / sigma2)
    i = np.arange(0, num_nodes).repeat(k)
    j = indices.reshape(num_nodes * k)
    v = dist.reshape(num_nodes * k)
    weights = sp.coo_matrix((v, (i, j)), shape=(num_nodes, num_nodes))
    # no self-loop
    weights.setdiag(0)
    # undirected graph
    bigger = weights.T > weights
    weights = weights - weights.multiply(bigger) + weights.T.multiply(bigger)
    return weights


def _get_normalize_adj(dist, indices):
    adj = _get_weights(dist, indices)
    adj = sp.coo_matrix(adj)
    row_sum = np.array(adj.sum(1))
    row_sum_nonzero = np.where(row_sum != 0, row_sum, 1)  # Avoid division by zero
    d_inv = np.power(row_sum_nonzero, -0.5).flatten()
    d_inv[np.isinf(d_inv)] = 0.0
    d_mat_inv_sqrt = sp.diags(d_inv)
    return adj.dot(d_mat_inv_sqrt).transpose().dot(d_mat_inv_sqrt).tocoo()


def build_graph(coordinates, k=4):
    """
    :param coordinates: positions for 3D point cloud (N * 3)
    :param k: number of nearest neighbors
    :return: adjacency matrix for 3D point cloud
    """
    from scipy.spatial import cKDTree
    tree = cKDTree(coordinates)
    dist, indices = tree.query(coordinates, k=k)
    return _get_normalize_adj(dist, indices)
```

```
def create_graph_dataset(indices_list, y, connectivity = 10):
    graph_dataset = []
    for i, points in enumerate(indices_list):
        c = build_graph(points, k=connectivity)
        #edge_index = build_graph(points, k = connectivity)
        edge_list = torch.from_numpy(np.vstack((c.row, c.col))).type(torch.long)
#         edge_index = torch.from_numpy(edge_index).type(torch.long)
        edge_weight = torch.from_numpy(c.data.reshape(-1, 1))
        labels = torch.tensor([int(y[i])], dtype=torch.long)

        # Convert COO matrix to tensor
    #     adj_tensor = torch.sparse_coo_tensor((c.row, c.col), c.data, c.shape).to_dense()
        data = Data(x=torch.from_numpy(features_list[i]), edge_index=edge_list, y=labels, edge_attr = edge_weight)
    # edge_attr=edge_weight
        graph_dataset.append(data)

    return graph_dataset



graph_dataset = create_graph_dataset(indices_list, y, connectivity = 8)
```

Having a scoop at the obtained list of `torch_geometric.data.data.Data` objects, with each Data object being populated with the features tensor as `data.x`, Adjacency list as `data.edge_index`, Edge Attributes, containing the distance metric as `data.edge_attr`, and the sparsely constructed adjacency matrix as `data.adj`.

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()
data = graph_dataset[0]
edge_tensor = data.edge_index
# Convert the edge tensor to a list of tuples
edge_list = [(edge_tensor[0, i].item(), edge_tensor[1, i].item()) for i in range(edge_tensor.shape[1])]


#print(data.edge_index.shape)
G.add_edges_from(edge_list)
pos = nx.spring_layout(G, iterations=15, seed=1721)
fig, ax = plt.subplots(figsize=(15, 9))
ax.axis("off")
plot_opt = {"node_size" : 10, "with_labels" : False, "width" : 0.05}
nx.draw_networkx(G, pos=pos, ax=ax, **plot_opt)
plt.show()

print(f'Number of graphs to work upon : {len(graph_dataset)}')

print(f'For the FIRST graph in the  graph dataset : ')
print(f'Type of each graph entity data object: {type(data)}')
print(f'Number of nodes: {data.num_nodes}')
print(f'Number of edges: {data.num_edges}')
print(f'Number of node features: {data.num_node_features}')
print(f'Number of edges features: {data.num_edge_features}')
#print(f'Number of edges features: {data.num_classes}')
```



```
Number of graphs to work upon : 16000
For the FIRST graph in the  graph dataset :
Type of each graph entity data object: <class 'torch_geometric.data.data.Data'>
Number of nodes: 1529
Number of edges: 12624
Number of node features: 3
Number of edges features: 1
```

## The SAGEConv Architecture : GraphSAGE

GraphSAGE is one of the most popular GNN architectures, utilizing neighborhood aggregations via the use of a more expressive convolutional operator.

```
import torch
from torch_geometric.nn import MessagePassing
from torch_geometric.utils import add_self_loops
class RandomSubsetConv(MessagePassing):
    def __init__(self, in_channels, out_channels, subset_size):
        super(RandomSubsetConv, self).__init__(aggr='add') # "add" aggregation.
        self.lin = torch.nn.Linear(in_channels, out_channels)
        self.subset_size = subset_size

    def forward(self, x, edge_index):
        # x has shape [N, in_channels]
        # edge_index has shape [2, E]

        # Step 1: Add self-loops to the adjacency matrix
        edge_index, _ = add_self_loops(edge_index, num_nodes=x.size(0))

        # Step 2: Sample a subset of neighbors for each node
        row, col = edge_index
        subset_indices = torch.randperm(row.size(0))[:self.subset_size]
        subset_edge_index = edge_index[:, subset_indices]

        # Step 3: Perform message passing
        return self.propagate(subset_edge_index, size=(x.size(0), x.size(0)), x=x)

    def message(self, x_j, edge_index, size):
        # x_j has shape [E, in_channels]
        # Step 4: Aggregate node features
        return self.lin(x_j)

    def update(self, aggr_out):
        # aggr_out has shape [N, out_channels]
        # Step 5: Update node features
        return aggr_out
```

```
import torch
import torch.utils.data as data
from torchvision import transforms
from torch_geometric.nn import GCNConv, GATConv, GATv2Conv, TransformerConv, global_mean_pool
from torch_geometric.nn.norm import BatchNorm
import torch.nn as nn


class GraphSAGE(torch.nn.Module):
    def __init__(self, c_in, c_hidden, c_out, dp_rate_linear=0.3, dp_rate_dropout=0.5, subset_size=10):
        super().__init__()
        torch.manual_seed(17)

        self.conv1 = RandomSubsetConv(c_in, c_hidden, subset_size = 4)
        self.norm1 = nn.BatchNorm1d(c_hidden)
        self.conv2 = RandomSubsetConv(c_hidden, 2 * c_hidden, subset_size = 4)
        self.norm2 = nn.BatchNorm1d(2 * c_hidden)
        self.conv3 = RandomSubsetConv(2 * c_hidden, c_hidden, subset_size = 4)
        self.norm3 = nn.BatchNorm1d(c_hidden)

        self.lin1 = nn.Linear(c_hidden, c_hidden // 2)
        self.lin2 = nn.Linear(c_hidden //2, c_out)

        self.dp_rate_linear = dp_rate_linear
        self.dp_rate_dropout = dp_rate_dropout
        self.dropout = nn.Dropout(dp_rate_dropout)

    def forward(self, x, edge_index, batch):
        x = self.conv1(x, edge_index)
        x = self.norm1(x)
        x = x.relu()
        x = self.dropout(x)

        x = self.conv2(x, edge_index)
        x = self.norm2(x)
        x = x.relu()
        x = self.dropout(x)

        x = self.conv3(x, edge_index)
        x = self.norm3(x)
        x = x.relu()
        x = self.dropout(x)

        x = global_mean_pool(x, batch)
        x = self.dropout(x)
        x = self.lin1(x)
        x = x.relu()
        x = self.lin2(x)

        return x
```