```
import torch
# the library for Machine Learning

import matplotlib.pyplot as plt
# draw image

from sklearn.manifold import TSNE
# a tool to visualize high-dimensional data

from torch_geometric.datasets import Planetoid
from torch_geometric.nn import Node2Vec
# torch_geometric (PyG) is a library for GNNs
# https://pytorch-geometric.readthedocs.io/en/latest/


dataset = Planetoid(root='/home/dw/Cora', name='Cora')
# derive a citation network
# change it to your path
# https://pytorch-
geometric.readthedocs.io/en/latest/generated/torch_geometric.datasets.Planetoid.html
data = dataset[0]

device = 'cuda' if torch.cuda.is_available() else 'cpu'


model = Node2Vec(data.edge_index, embedding_dim=128, walk_length=20,
                 context_size=10, walks_per_node=10, num_negative_samples=1,
                 sparse=True).to(device)
loader = model.loader(batch_size=128, shuffle=True, num_workers=4)
# shuffle: randomize data for each epoch

optimizer = torch.optim.SparseAdam(model.parameters(), lr=0.01)
# define the optimizer method
# SGD is one of the popular optimizer

def train():
    model.train()
    total_loss = 0
    for pos_rw, neg_rw in loader:
        optimizer.zero_grad()
        # clear gradient values in cache
        loss = model.loss(pos_rw.to(device), neg_rw.to(device))
        # build-in loss function for the model
        loss.backward()
        # compute gradient
        optimizer.step()
        # update parameters
        total_loss += loss.item()
    return total_loss / len(loader)


@torch.no_grad()
def test():
    model.eval()
    z = model()
    acc = model.test(z[data.train_mask], data.y[data.train_mask],
                     z[data.test_mask], data.y[data.test_mask], max_iter=150)
    return acc


for epoch in range(1, 51):
    loss = train()
    acc = test()
    print(f'Epoch: {epoch:02d}, Loss: {loss:.4f}, Acc: {acc:.4f}')


# visualize nodes
@torch.no_grad()
def plot_points(colors):
```

```
    model.eval()
    z = model(torch.arange(data.num_nodes, device=device))
    z = TSNE(n_components=2).fit_transform(z.cpu().numpy())
    y = data.y.cpu().numpy()

    plt.figure(figsize=(8, 8))
    for i in range(dataset.num_classes):
        plt.scatter(z[y == i, 0], z[y == i, 1], s=20, color=colors[i])
    plt.axis('off')
    plt.savefig('./output.jpg')
    plt.show()


colors = ['#d00000', '#ffba08', '#3f88c5', '#bdb2ff', '#136f63', '#ED9E6F', '#B2DB81']
plot_points(colors)
```