```
! pip install torch_geometric
```

```
Collecting torch_geometric
  Downloading torch_geometric-2.6.1-py3-none-any.whl.metadata (63 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 63.1/63.1 kB 2.6 MB/s eta
0:00:00
ent already satisfied: aiohttp in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (3.11.12)
Requirement already satisfied: fsspec in
/usr/local/lib/python3.10/dist-packages (from torch_geometric)
(2024.12.0)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch_geometric) (3.1.4)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from torch_geometric)
(1.26.4)
Requirement already satisfied: psutil>=5.8.0 in
/usr/local/lib/python3.10/dist-packages (from torch_geometric) (5.9.5)
Requirement already satisfied: pyparsing in
/usr/local/lib/python3.10/dist-packages (from torch_geometric) (3.2.0)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from torch_geometric)
(2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from torch_geometric) (4.67.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (2.4.6)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (1.3.2)
Requirement already satisfied: async-timeout<6.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (5.0.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (25.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp-
>torch_geometric) (1.18.3)
Requirement already satisfied: MarkupSafe>=2.0 in
```

```
/usr/local/lib/python3.10/dist-packages (from jinja2->torch_geometric)
(3.0.2)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.10/dist-packages (from numpy->torch_geometric)
(1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.10/dist-packages (from numpy->torch_geometric)
(1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.10/dist-packages (from numpy->torch_geometric)
(0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-
packages (from numpy->torch_geometric) (2025.0.1)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.10/dist-packages (from numpy->torch_geometric)
(2022.0.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.10/dist-packages (from numpy->torch_geometric)
(2.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests-
>torch_geometric) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests-
>torch_geometric) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests-
>torch_geometric) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests-
>torch_geometric) (2025.1.31)
Requirement already satisfied: typing-extensions>=4.1.0 in
/usr/local/lib/python3.10/dist-packages (from multidict<7.0,>=4.5-
>aiohttp->torch_geometric) (4.12.2)
Requirement already satisfied: intel-openmp>=2024 in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy-
>torch_geometric) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy-
>torch_geometric) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.10/dist-packages (from tbb==2022.*->mkl->numpy-
>torch_geometric) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.10/dist-packages (from mkl_umath->numpy-
>torch_geometric) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.10/dist-packages (from intel-openmp>=2024->mkl-
>numpy->torch_geometric) (2024.2.0)
```

```
Downloading torch_geometric-2.6.1-py3-none-any.whl (1.1 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 31.7 MB/s eta
0:00:0000:01
etric
Successfully installed torch_geometric-2.6.1
```

Importing Essential Libraries

```python
import os
import torch
import warnings
import h5py
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import kneighbors_graph
from torch.nn import Linear
import torch.nn as nn
from torch_geometric.data import Data, Batch
from torch_geometric.loader import DataLoader
import torch.optim as optim
import pytorch_lightning as pl
import torch.nn.functional as F
from torch_geometric.nn import SAGEConv
from torch_geometric.nn import global_mean_pool

warnings.filterwarnings("ignore")

CONFIG = {
    'data_path': '/kaggle/input/datasettt/Quark Gluon Data Set.hdf5',
    'max_samples': 20000,
    'n_neighbors': 2,
    'train_size': 8000,
    'test_size': 1000,
    'batch_size': 32,
    'hidden_dim': 32,
    'dropout_rate': 0.3,
    'learning_rate': 1e-3,
    'max_epochs': 40,
    'seed': 17
}
```

Load Data

```python
def load_data(file_path, max_samples):
    with h5py.File(file_path, 'r') as f:
        X_jets = np.array(f['X_jets'][:max_samples])
        labels = np.array(f['y'][:max_samples])
    return X_jets, labels
```

```python
X_jets, labels = load_data(CONFIG['data_path'], CONFIG['max_samples'])
print(f"Loaded dataset with shape: {X_jets.shape}")
```

```
Loaded dataset with shape: (20000, 125, 125, 3)
```

```python
def create_graph_dataset(data, labels, n_neighbors=2):
    reshaped_data = data.reshape((-1, data.shape[1]*data.shape[2], 3))
    node_list = []

    for i, x in enumerate(reshaped_data):
        non_black_pixels = np.any(x != [0., 0., 0.], axis=-1)
        node_list.append(x[non_black_pixels])

    dataset = []
    for i, nodes in enumerate(node_list):
        edges = kneighbors_graph(nodes, n_neighbors,
mode='connectivity', include_self=True)
        c = edges.tocoo()

        edge_index = torch.from_numpy(np.vstack((c.row,
c.col))).type(torch.long)
        edge_attr = torch.from_numpy(c.data.reshape(-1, 1))
        y = torch.tensor([int(labels[i])], dtype=torch.long)

        graph = Data(x=torch.from_numpy(nodes).float(),
                     edge_index=edge_index,
                     edge_attr=edge_attr,
                     y=y)
        dataset.append(graph)

    return dataset

dataset = create_graph_dataset(X_jets, labels, CONFIG['n_neighbors'])

train_loader = DataLoader(dataset[:CONFIG['train_size']],
                          batch_size=CONFIG['batch_size'],
                          shuffle=True)
test_loader =
DataLoader(dataset[CONFIG['train_size']:CONFIG['train_size']
+CONFIG['test_size']],
                         batch_size=CONFIG['batch_size'],
                         shuffle=False)
val_loader = DataLoader(dataset[CONFIG['train_size']
+CONFIG['test_size']:],
                        batch_size=CONFIG['batch_size'],
                        shuffle=False)

#print dataset information
data_sample = dataset[0]
print(f'Number of nodes: {data_sample.num_nodes}')
print(f'Number of edges: {data_sample.num_edges}')
print(f'Number of node features: {data_sample.num_node_features}')
print(f'Number of edge features: {data_sample.num_edge_features}')
print(f'Sample graph: {data_sample}')

print(f'Number of batches: Train={len(train_loader)},
Test={len(test_loader)}, Val={len(val_loader)}')
```

```
Number of nodes: 884
Number of edges: 1768
Number of node features: 3
Number of edge features: 1
Sample graph: Data(x=[884, 3], edge_index=[2, 1768], edge_attr=[1768,
1], y=[1])
Number of batches: Train=250, Test=32, Val=344
```

Building and Training the Model

```
class GraphSAGEModel(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels,
dropout_rate=0.3):
        super().__init__()
        torch.manual_seed(CONFIG['seed'])

        self.conv1 = SAGEConv(in_channels, hidden_channels)
        self.conv2 = SAGEConv(hidden_channels, 2*hidden_channels)
        self.conv3 = SAGEConv(2*hidden_channels, 4*hidden_channels)

        self.lin1 = Linear(4*hidden_channels, 32*out_channels)
        self.lin2 = Linear(32*out_channels, 8*out_channels)
        self.lin3 = Linear(8*out_channels, out_channels)

        self.dropout_rate = dropout_rate

    def forward(self, x, edge_index, batch):
        x = F.relu(self.conv1(x, edge_index))
        x = F.relu(self.conv2(x, edge_index))
        x = F.relu(self.conv3(x, edge_index))

        x = global_mean_pool(x, batch)

        x = F.dropout(x, p=self.dropout_rate, training=self.training)
        x = F.relu(self.lin1(x))
        x = F.dropout(x, p=self.dropout_rate, training=self.training)
        x = F.relu(self.lin2(x))
        x = self.lin3(x)

        return x
```

```
class GNNClassifier(pl.LightningModule):
    def __init__(self, in_channels, hidden_channels, out_channels,
dropout_rate=0.3, lr=1e-3):
        super().__init__()
        self.save_hyperparameters()

        self.model = GraphSAGEModel(in_channels, hidden_channels,
out_channels, dropout_rate)
        self.loss_fn = nn.BCEWithLogitsLoss() if out_channels == 1
else nn.CrossEntropyLoss()

    def forward(self, data):
        x, edge_index, batch_idx = data.x, data.edge_index, data.batch
        return self.model(x, edge_index, batch_idx)

    def step(self, batch, mode="train"):
        logits = self(batch)

        if self.hparams.out_channels == 1:
            y = batch.y.float()
            preds = (logits > 0).float()
        else:
            y = batch.y.squeeze()
            preds = logits.argmax(dim=-1)

        loss = self.loss_fn(logits, y)
        acc = (preds == y).sum().float() / preds.shape[0]

        self.log(f'{mode}_loss', loss, prog_bar=True)
        self.log(f'{mode}_acc', acc, prog_bar=True)

        return loss


    def training_step(self, batch, batch_idx):
        return self.step(batch, mode="train")

    def validation_step(self, batch, batch_idx):
        self.step(batch, mode="val")

    def test_step(self, batch, batch_idx):
        self.step(batch, mode="test")

    def configure_optimizers(self):
        return optim.Adam(self.parameters(), lr=self.hparams.lr)

def train_model():
    pl.seed_everything(CONFIG['seed'])

    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    print(f"Using device: {device}")

    model = GNNClassifier(
        in_channels=3,
        hidden_channels=CONFIG['hidden_dim'],
        out_channels=2,  #binary classification
        dropout_rate=CONFIG['dropout_rate'],
        lr=CONFIG['learning_rate']
    )

    trainer = pl.Trainer(
        max_epochs=CONFIG['max_epochs'],
        accelerator='gpu' if torch.cuda.is_available() else 'cpu',
        devices=1
    )

    trainer.fit(model, train_loader, val_loader)

    best_model_path = trainer.checkpoint_callback.best_model_path
    if best_model_path:
        model = GNNClassifier.load_from_checkpoint(best_model_path)

    val_result = trainer.test(model, dataloaders=val_loader,
verbose=False)
    test_result = trainer.test(model, dataloaders=test_loader,
verbose=False)

    results = {
        "validation_accuracy": val_result[0]['test_acc'],
        "test_accuracy": test_result[0]['test_acc']
    }

    return trainer, model, results

trainer, model, results = train_model()
print(f"Model architecture:\n{model}")
```

```
Using device: cuda
```

```
{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"7f62450c6b834effa66853c35debd84f","version_major":2,"vers
ion_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}
```

```
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
```

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"1726dcd8219b435da5adbbe72de79287","version_major":2,"version_minor":0}

{"model_id":"692c8773377e4b58a399fc63a41ce1d4","version_major":2,"version_minor":0}

```
Model architecture:
GNNClassifier(
  (model): GraphSAGEModel(
    (conv1): SAGEConv(3, 32, aggr=mean)
    (conv2): SAGEConv(32, 64, aggr=mean)
    (conv3): SAGEConv(64, 128, aggr=mean)
    (lin1): Linear(in_features=128, out_features=64, bias=True)
    (lin2): Linear(in_features=64, out_features=16, bias=True)
    (lin3): Linear(in_features=16, out_features=2, bias=True)
  )
  (loss_fn): CrossEntropyLoss()
)
```

```
print(f"Results: {results}")
```

```
Results: {'validation_accuracy': 0.7027001976966858, 'test_accuracy':
0.7250475287437439}
```

```
dataset = create_graph_dataset(X_jets, labels, 10)
```

```
trainer, model, results = train_model()
print(f"Model architecture:\n{model}")
```

```
Using device: cuda
```

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"7cc4b846c4574019a7597ca1e5720501","version_major":2,"version_minor":0}

```
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
{"model_id":"","version_major":2,"version_minor":0}
```

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"98618d5496b74415af9648d03f12ef9d","version_major":2,"version_minor":0}

{"model_id":"3890a460b7ba4cc192110e65cda27a51","version_major":2,"version_minor":0}

```
Model architecture:
GNNClassifier(
  (model): GraphSAGEModel(
    (conv1): SAGEConv(3, 32, aggr=mean)
    (conv2): SAGEConv(32, 64, aggr=mean)
    (conv3): SAGEConv(64, 128, aggr=mean)
    (lin1): Linear(in_features=128, out_features=64, bias=True)
    (lin2): Linear(in_features=64, out_features=16, bias=True)
    (lin3): Linear(in_features=16, out_features=2, bias=True)
  )
  (loss_fn): CrossEntropyLoss()
)
```

```python
print(f"Results: {results}")
```

```
Results: {'validation_accuracy': 0.6972792744636536, 'test_accuracy':
0.7086634635925293}
```

```python
dataset = create_graph_dataset(X_jets, labels, 5)
```

```python
trainer, model, results = train_model()
print(f"Model architecture:\n{model}")
```

```
Using device: cuda

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"1035252f9bc34c8db9e6a8e3654352a1","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}
```

```
{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

{"model_id":"2c4a4f98bc7f4a00ae2a0d216abfa19b","version_major":2,"vers
ion_minor":0}

{"model_id":"d35d56d0fdb546969cccbd2ac06f894e","version_major":2,"vers
ion_minor":0}

Model architecture:
GNNClassifier(
  (model): GraphSAGEModel(
    (conv1): SAGEConv(3, 32, aggr=mean)
    (conv2): SAGEConv(32, 64, aggr=mean)
    (conv3): SAGEConv(64, 128, aggr=mean)
    (lin1): Linear(in_features=128, out_features=64, bias=True)
    (lin2): Linear(in_features=64, out_features=16, bias=True)
    (lin3): Linear(in_features=16, out_features=2, bias=True)
  )
  (loss_fn): CrossEntropyLoss()
)

print(f"Results: {results}")
```

```
Results: {'validation_accuracy': 0.6976399421691895, 'test_accuracy':
0.7167098522186279}
```

Key Observations

Performance Comparison. Here's a summary of our results:

k value	Validation Accuracy	Test Accuracy
2	70.27%	72.50%
5	69.76%	71.67%
10	69.73%	70.87%

- Inverse Relationship: There appears to be an inverse relationship between the number of neighbors (k) and model performance. As k increases, both validation and test accuracy tend to decrease.
- Best Performance: The model with k=2 achieved the best performance with 72.50% test accuracy, outperforming the other configurations by a noticeable margin.
- Validation-Test Gap: All three models show a positive gap between validation and test accuracy (test accuracy is higher), which suggests good generalization rather than overfitting.

- From this we can understand that most of the discriminative information is contained in nearest-neighbor relationships
- In particle jet classification, the most relevant information might be contained in the closest spatial relationships. As we look more distant neighbors, we may be introducing noise rather than useful signal.
- This is also good considering the computational efficiency, since small k model would be more computationally efficient since it creates graphs only with fewer edges.