gnn

March 25, 2024

1 DeepFalcon

1.1 Common Task 2. Jets as graphs

- Please choose a graph-based GNN model of your choice to classify (quark/gluon) jets. Proceed as follows:
  1. Convert the images into a point cloud dataset by only considering the non-zero pixels for every event.
  2. Cast the point cloud data into a graph representation by coming up with suitable representations for nodes and edges.
  3. Train your model on the obtained graph representations of the jet events.
- Discuss the resulting performance of the chosen architecture.

2 Genie

2.1 Common Task 2. Jets as graphs

- Please choose a graph-based GNN model of your choice to classify (quark/gluon) jets. Proceed as follows:
  1. Convert the images into a point cloud dataset by only considering the non-zero pixels for every event.
  2. Cast the point cloud data into a graph representation by coming up with suitable representations for nodes and edges.
  3. Train your model on the obtained graph representations of the jet events.
- Discuss the resulting performance of the chosen architecture.

[ ]:
```
import torch
import numpy as np
import h5py
import os
import matplotlib.pyplot as plt
import torch.nn as nn
import torch.nn.functional as F
from tqdm.autonotebook import tqdm
import torchvision
import random
import cv2
import torch_geometric
```

```
from torch_geometric.nn import␣
  ↪GCNConv,global_mean_pool,GATConv,SAGEConv,GraphConv
from torch_geometric.data import Data, Batch
from torch_geometric.loader import DataLoader
from sklearn.neighbors import kneighbors_graph
```

```
/tmp/ipykernel_44548/2131603270.py:8: TqdmWarning: IProgress not found. Please
update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from tqdm.autonotebook import tqdm
```

2.2 SEEDING - For Reproducability

[ ]:
```
seed = 0
random.seed(seed)
np.random.seed(seed)
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic = True
```

2.3 Data Preprocessing

[ ]:
```
data_path = 'quark-gluon_data-set_n139306.hdf5'
num_samples = 15000

x_jets = np.array(h5py.File(data_path,'r')['X_jets'][:num_samples])
labels  = np.array(h5py.File(data_path,'r')['y'][:num_samples])
```

[ ]:
```
x_jets.shape
```

[ ]:
```
(15000, 125, 125, 3)
```

[ ]:
```
def point_cloud(x_jets):
    point_clouds = []
    for img in x_jets:
        nonzero_coords = np.any(img!=[0,0,0],axis=-1)
        values = img[nonzero_coords]
        point_clouds.append(values)
    return point_clouds

def graph_representation(point_clouds,n_neighbor = 10):
    graph_representation = []
    for i,point_cloud in enumerate(point_clouds):
        edges =␣
  ↪kneighbors_graph(point_cloud,n_neighbors=n_neighbor,mode='connectivity')

        edges = edges.tocoo()
        edge_index = torch.tensor(np.vstack((edges.row,edges.col))).type(torch.
↪long)
        edge_attr = torch.tensor(edges.data.reshape(-1,1))
        label  = torch.tensor(int(labels[i]),dtype=torch.long)
        data = torch_geometric.data.Data(x = torch.tensor(point_cloud),␣
↪edge_index = edge_index, edge_attr=edge_attr,y=label)
        graph_representation.append(data)

    return graph_representation
```

[ ]:
```
point_clouds = point_cloud(x_jets)
dataset = graph_representation(point_clouds)
```

[ ]:
```
train_dataset = dataset[:10000]
val_dataset = dataset[10000:12000]
test_dataset = dataset[12000:]

batch_size = 32
train_dataloader = DataLoader(train_dataset,batch_size=batch_size,shuffle=True)
val_dataloader = DataLoader(val_dataset,batch_size=batch_size,shuffle=False)
test_dataloader = DataLoader(test_dataset,batch_size=batch_size,shuffle=False)
```

[ ]:
```
data = next(iter(train_dataloader))[0]
```

[ ]:
```
len(train_dataloader.dataset)
```

[ ]:
```
10000
```

[ ]:
```
print(data)
print('==============================================================')

# Gather some statistics about the first graph.
print(f'number of feature: {data.num_features}')
print(f'Number of nodes: {data.num_nodes}')
print(f'Number of edges: {data.num_edges}')
print(f'Average node degree: {data.num_edges / data.num_nodes:.2f}')
print(f'Has isolated nodes: {data.has_isolated_nodes()}')
print(f'Has self-loops: {data.has_self_loops()}')
print(f'Is undirected: {data.is_undirected()}')
```

```
Data(x=[330, 3], edge_index=[2, 3300], edge_attr=[3300, 1], y=[1])
==============================================================
number of feature: 3
Number of nodes: 330
Number of edges: 3300
Average node degree: 10.00
Has isolated nodes: False
Has self-loops: False
Is undirected: False
```

[ ]:
```
# for step,data in enumerate(train_dataloader):
#     print(f'Step {step + 1}:')
#     print('=======')
#     print(f'Number of graphs in the current batch: {data.num_graphs}')
#     print(data)
#     print()
```

3 MODEL

3.1 1) Graph Convolutional Networks (GCN)

[ ]:
```
class GCN(nn.Module):
    def __init__(self,in_channels ,hidden_dim, output_dim,p=0.3):
        super().__init__()
        self.conv1 = GCNConv(in_channels,hidden_dim)
        self.conv2 = GCNConv(hidden_dim,hidden_dim)
        self.conv3 = GCNConv(hidden_dim,hidden_dim)
        self.fc1 = nn.Linear(hidden_dim,hidden_dim*2)
        self.fc2= nn.Linear(hidden_dim*2,output_dim)
        self.p = p
    def forward(self,data):
        x, edge_index, edge_attr,batch = data.x.float(), data.edge_index, data.
 ↪edge_attr.float(),data.batch
        x = self.conv1(x,edge_index)
        x = F.relu(x)
        x = F.dropout(x,p=self.p,training=self.training)
        x = self.conv2(x,edge_index)
        x = F.relu(x)
        x = F.dropout(x,p=self.p,training=self.training)
        x = self.conv3(x,edge_index)

        x = global_mean_pool(x,batch) #[batch_size,hidden_channels]

        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc1(x)
        x = F.relu(x)
        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc2(x)
        return x
```

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = GCN(in_channels=3,hidden_dim=128,output_dim=2).to(device)
optimizer = torch.optim.Adam(model.parameters(),lr=3e-4)
criterion = nn.CrossEntropyLoss()
```

```
[ ]: for epoch in range(30):
         model.train()
         train_loss = 0
         train_correct = 0
         for train_data in train_dataloader:
             train_data = train_data.to(device)
             optimizer.zero_grad()
             train_output = model(train_data)
             loss = criterion(train_output, train_data.y)
             loss.backward()
             optimizer.step()

         # Validation
         model.eval()
         val_loss = 0
         val_correct = 0
         for val_data in val_dataloader:
             val_data = val_data.to(device)
             val_output = model(val_data)
             val_loss += criterion(val_output, val_data.y).item()
             val_pred = val_output.argmax(dim=1)
             # print(len(pred))
             val_correct += int((val_pred==val_data.y).sum())
             # val_correct += (pred == val_data.y).sum().float()/pred.shape[0]


         val_loss /= len(val_dataloader)
         # print(f'val_correct{val_correct}')
         val_acc = val_correct / len(val_dataloader.dataset)

         print(f'Epoch: {epoch+1}, Train Loss: {loss:.4f} Val Loss: {val_loss:.4f},␣
      ↪Val Acc: {val_acc:.4f}')
```

```
Epoch: 1, Train Loss: 0.6863 Val Loss: 0.6412, Val Acc: 0.6770
Epoch: 2, Train Loss: 0.5163 Val Loss: 0.6225, Val Acc: 0.6975
Epoch: 3, Train Loss: 0.7050 Val Loss: 0.6167, Val Acc: 0.6965
Epoch: 4, Train Loss: 0.5615 Val Loss: 0.6164, Val Acc: 0.6880
Epoch: 5, Train Loss: 0.5614 Val Loss: 0.6077, Val Acc: 0.6995
Epoch: 6, Train Loss: 0.4354 Val Loss: 0.6053, Val Acc: 0.7010
Epoch: 7, Train Loss: 0.5993 Val Loss: 0.6023, Val Acc: 0.6990
Epoch: 8, Train Loss: 0.5784 Val Loss: 0.6017, Val Acc: 0.7010
Epoch: 9, Train Loss: 0.6372 Val Loss: 0.5990, Val Acc: 0.7025
Epoch: 10, Train Loss: 0.6017 Val Loss: 0.5964, Val Acc: 0.6990
Epoch: 11, Train Loss: 0.5384 Val Loss: 0.5995, Val Acc: 0.6955
Epoch: 12, Train Loss: 0.5065 Val Loss: 0.5956, Val Acc: 0.6970
Epoch: 13, Train Loss: 0.5417 Val Loss: 0.5929, Val Acc: 0.6990
```

```
Epoch: 14, Train Loss: 0.4236 Val Loss: 0.5964, Val Acc: 0.6980
Epoch: 15, Train Loss: 0.4347 Val Loss: 0.6050, Val Acc: 0.6825
Epoch: 16, Train Loss: 0.7658 Val Loss: 0.5942, Val Acc: 0.6985
Epoch: 17, Train Loss: 0.9048 Val Loss: 0.5902, Val Acc: 0.6945
Epoch: 18, Train Loss: 0.6492 Val Loss: 0.5905, Val Acc: 0.6945
Epoch: 19, Train Loss: 0.5139 Val Loss: 0.5996, Val Acc: 0.6845
Epoch: 20, Train Loss: 0.8060 Val Loss: 0.5904, Val Acc: 0.6980
Epoch: 21, Train Loss: 0.4595 Val Loss: 0.5888, Val Acc: 0.6970
Epoch: 22, Train Loss: 0.6489 Val Loss: 0.5904, Val Acc: 0.6970
Epoch: 23, Train Loss: 0.6757 Val Loss: 0.5975, Val Acc: 0.6920
Epoch: 24, Train Loss: 0.5330 Val Loss: 0.5962, Val Acc: 0.6850
Epoch: 25, Train Loss: 0.5630 Val Loss: 0.5886, Val Acc: 0.6970
Epoch: 26, Train Loss: 0.5413 Val Loss: 0.5885, Val Acc: 0.7005
Epoch: 27, Train Loss: 0.6915 Val Loss: 0.5877, Val Acc: 0.6975
Epoch: 28, Train Loss: 0.7688 Val Loss: 0.5910, Val Acc: 0.6950
Epoch: 29, Train Loss: 0.3441 Val Loss: 0.5897, Val Acc: 0.6970
Epoch: 30, Train Loss: 0.6735 Val Loss: 0.5887, Val Acc: 0.6950
```

[ ]:
```
model.eval()
test_correct = 0
for data in test_dataloader:
    data = data.to(device)
    output = model(data)
    pred = output.argmax(dim=1)
    test_correct += int((pred==data.y).sum())

test_acc = test_correct / len(test_dataloader.dataset)
print(f'Test Accuracy: {test_acc:.4f}')
```

```
Test Accuracy: 0.6947
```

[ ]:
```
#test_accuracy  = 0.6887, valid_accuracy = 0.6865 for n_neigbors = 2
#test_accuracy  = 0.6943, valid_accuracy = 0.6885 for n_neigbors = 5
#test_accuracy  = 0.6947, valid_accuracy = 0.6950 for n_neigbors = 10
```

3.2 2) Graph Attention Networks (GAT)

[ ]:
```
class GAT(nn.Module):
    def __init__(self,in_channels ,hidden_dim, output_dim,p=0.3):
        super().__init__()
        self.attn1 = GATConv(in_channels,hidden_dim)
        self.attn2 = GATConv(hidden_dim,hidden_dim)
        self.attn3 = GATConv(hidden_dim,hidden_dim)
        self.fc1 = nn.Linear(hidden_dim,hidden_dim)
        self.fc2= nn.Linear(hidden_dim,output_dim)
        self.p = p
    def forward(self,data):
        x, edge_index, edge_attr,batch = data.x.float(), data.edge_index, data.
↪edge_attr.float(),data.batch
        x = self.attn1(x,edge_index, edge_attr = edge_attr)
        x = F.relu(x)
        # x = F.dropout(x,p=self.p,training=self.training)
        x = self.attn2(x,edge_index,edge_attr = edge_attr)
        x = F.relu(x)
        # x = F.dropout(x,p=self.p,training=self.training)
        x = self.attn3(x,edge_index,edge_attr = edge_attr)

        x = global_mean_pool(x,batch)

        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc1(x)
        x = F.relu(x)
        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc2(x)
        return x
```

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = GAT(in_channels=3,hidden_dim=128,output_dim=2).to(device)
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
criterion = nn.CrossEntropyLoss()
```

[ ]:
```
for epoch in range(30):
    model.train()

    for train_data in train_dataloader:
        train_data = train_data.to(device)
        optimizer.zero_grad()
        train_output = model(train_data)
        loss = criterion(train_output, train_data.y)
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    val_loss = 0
    val_correct = 0
    for val_data in val_dataloader:
        val_data = val_data.to(device)
        val_output = model(val_data)
        val_loss += criterion(val_output, val_data.y).item()
        pred = val_output.argmax(dim=1)
        # print(len(pred))
        val_correct += int((pred==val_data.y).sum())
        # val_correct += (pred == val_data.y).sum().float()/pred.shape[0]
    val_loss /= len(val_dataloader)
    # print(f'val_correct{val_correct}')
    val_acc = val_correct / len(val_dataloader.dataset)

    print(f'Epoch: {epoch+1}, Train Loss: {loss:.4f} Val Loss: {val_loss:.4f},␣
↪Val Acc: {val_acc:.4f}')
```

```
Epoch: 1, Train Loss: 0.4588 Val Loss: 0.6330, Val Acc: 0.6980
Epoch: 2, Train Loss: 0.4638 Val Loss: 0.6370, Val Acc: 0.6985
Epoch: 3, Train Loss: 0.5474 Val Loss: 0.6122, Val Acc: 0.6865
Epoch: 4, Train Loss: 0.6831 Val Loss: 0.6140, Val Acc: 0.6970
Epoch: 5, Train Loss: 0.6498 Val Loss: 0.6037, Val Acc: 0.6920
Epoch: 6, Train Loss: 0.7989 Val Loss: 0.6030, Val Acc: 0.6940
Epoch: 7, Train Loss: 0.5855 Val Loss: 0.6046, Val Acc: 0.7020
Epoch: 8, Train Loss: 0.6341 Val Loss: 0.5980, Val Acc: 0.6930
Epoch: 9, Train Loss: 0.6395 Val Loss: 0.5951, Val Acc: 0.7020
Epoch: 10, Train Loss: 0.5261 Val Loss: 0.5917, Val Acc: 0.7035
Epoch: 11, Train Loss: 0.5162 Val Loss: 0.5881, Val Acc: 0.7010
Epoch: 12, Train Loss: 0.6346 Val Loss: 0.5878, Val Acc: 0.6955
Epoch: 13, Train Loss: 0.6418 Val Loss: 0.5955, Val Acc: 0.6920
Epoch: 14, Train Loss: 0.6222 Val Loss: 0.5826, Val Acc: 0.6995
Epoch: 15, Train Loss: 0.7504 Val Loss: 0.5998, Val Acc: 0.6905
Epoch: 16, Train Loss: 0.5288 Val Loss: 0.5859, Val Acc: 0.6980
Epoch: 17, Train Loss: 0.4892 Val Loss: 0.5970, Val Acc: 0.6900
Epoch: 18, Train Loss: 0.6313 Val Loss: 0.5867, Val Acc: 0.7015
Epoch: 19, Train Loss: 0.4507 Val Loss: 0.5844, Val Acc: 0.6990
Epoch: 20, Train Loss: 0.7316 Val Loss: 0.5834, Val Acc: 0.6940
Epoch: 21, Train Loss: 0.4980 Val Loss: 0.5895, Val Acc: 0.6970
Epoch: 22, Train Loss: 0.7068 Val Loss: 0.5811, Val Acc: 0.6970
Epoch: 23, Train Loss: 0.6937 Val Loss: 0.5856, Val Acc: 0.6975
Epoch: 24, Train Loss: 0.5826 Val Loss: 0.6015, Val Acc: 0.6965
Epoch: 25, Train Loss: 0.5293 Val Loss: 0.5853, Val Acc: 0.6990
Epoch: 26, Train Loss: 0.3775 Val Loss: 0.5837, Val Acc: 0.7040
Epoch: 27, Train Loss: 0.7140 Val Loss: 0.5850, Val Acc: 0.6970
Epoch: 28, Train Loss: 0.7173 Val Loss: 0.5843, Val Acc: 0.6985
Epoch: 29, Train Loss: 0.8186 Val Loss: 0.5845, Val Acc: 0.7005
Epoch: 30, Train Loss: 0.7919 Val Loss: 0.5817, Val Acc: 0.6995
```

[ ]:
```
model.eval()
test_correct = 0
for data in test_dataloader:
    data = data.to(device)
    output = model(data)
    pred = output.argmax(dim=1)
    test_correct += int((pred==data.y).sum())

test_acc = test_correct / len(test_dataloader.dataset)
print(f'Test Accuracy: {test_acc:.4f}')
```

```
Test Accuracy: 0.6917
```

[ ]:
```
# test_accuracy  = 0.6870,valid_accuracy = 0.6955 for n_neigbors = 2
# test_accuracy  = 0.6917,valid_accuracy = 0.6980 for n_neigbors = 5
# test_accuracy  = 0.6950,valid_accuracy = 0.7060 for n_neigbors = 10

# test_accuracy  = 0.6917,valid_accuracy = 0.6995 for n_neigbors = 10 with␣
↪edge_attr present.
```

3.3 3) SageConv

[ ]:
```
class GraphSage(nn.Module):
    def __init__(self,in_channels ,hidden_dim, output_dim,p=0.3):
        super().__init__()
        self.attn1 = SAGEConv(in_channels,hidden_dim)
        self.attn2 = SAGEConv(hidden_dim,hidden_dim)
        self.attn3 = SAGEConv(hidden_dim,hidden_dim)
        self.fc1= nn.Linear(hidden_dim,hidden_dim*2)
        self.fc2= nn.Linear(hidden_dim*2,output_dim)
        self.p = p
    def forward(self,data):
        x, edge_index, edge_attr,batch = data.x.float(), data.edge_index, data.
↪edge_attr.float(),data.batch
        x = self.attn1(x,edge_index)
        x = F.relu(x)
        # x = F.dropout(x,p=self.p,training=self.training)
        x = self.attn2(x,edge_index)
        x = F.relu(x)
        # x = F.dropout(x,p=self.p,training=self.training)
        x = self.attn3(x,edge_index)

        x = global_mean_pool(x,batch)

        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc1(x)
        x = F.relu(x)
        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc2(x)
        return x
```

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = GraphSage(in_channels=3,hidden_dim=128,output_dim=2).to(device)
optimizer = torch.optim.Adam(model.parameters(),lr=3e-4)
criterion = nn.CrossEntropyLoss()
```

[ ]:
```
for epoch in range(30):
    model.train()

    for train_data in train_dataloader:
        train_data = train_data.to(device)
        optimizer.zero_grad()
        train_output = model(train_data)
        loss = criterion(train_output, train_data.y)
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    val_loss = 0
    val_correct = 0
    for val_data in val_dataloader:
        val_data = val_data.to(device)
        val_output = model(val_data)
        val_loss += criterion(val_output, val_data.y).item()
        pred = val_output.argmax(dim=1)
        # print(len(pred))
        val_correct += int((pred==val_data.y).sum())
        # val_correct += (pred == val_data.y).sum().float()/pred.shape[0]


    val_loss /= len(val_dataloader)
    # print(f'val_correct{val_correct}')
    val_acc = val_correct / len(val_dataloader.dataset)

    print(f'Epoch: {epoch+1}, Train Loss: {loss:.4f} Val Loss: {val_loss:.4f},␣
 ↪Val Acc: {val_acc:.4f}')
```

```
Epoch: 1, Train Loss: 0.6685 Val Loss: 0.6915, Val Acc: 0.5030
Epoch: 2, Train Loss: 0.6329 Val Loss: 0.6222, Val Acc: 0.6990
Epoch: 3, Train Loss: 0.4058 Val Loss: 0.6138, Val Acc: 0.6865
Epoch: 4, Train Loss: 0.7051 Val Loss: 0.6017, Val Acc: 0.7035
Epoch: 5, Train Loss: 0.7029 Val Loss: 0.6215, Val Acc: 0.6725
Epoch: 6, Train Loss: 0.5701 Val Loss: 0.5905, Val Acc: 0.7060
Epoch: 7, Train Loss: 0.6021 Val Loss: 0.5999, Val Acc: 0.6945
Epoch: 8, Train Loss: 0.5139 Val Loss: 0.5835, Val Acc: 0.7120
Epoch: 9, Train Loss: 0.6240 Val Loss: 0.5852, Val Acc: 0.7055
```

```
Epoch: 10, Train Loss: 0.6043 Val Loss: 0.5853, Val Acc: 0.7075
Epoch: 11, Train Loss: 0.6594 Val Loss: 0.5897, Val Acc: 0.7090
Epoch: 12, Train Loss: 0.5174 Val Loss: 0.5847, Val Acc: 0.7105
Epoch: 13, Train Loss: 0.7411 Val Loss: 0.5864, Val Acc: 0.7120
Epoch: 14, Train Loss: 0.6995 Val Loss: 0.5915, Val Acc: 0.6980
Epoch: 15, Train Loss: 0.3893 Val Loss: 0.5937, Val Acc: 0.7060
Epoch: 16, Train Loss: 0.5466 Val Loss: 0.5812, Val Acc: 0.7090
Epoch: 17, Train Loss: 0.7335 Val Loss: 0.5847, Val Acc: 0.7075
Epoch: 18, Train Loss: 0.4805 Val Loss: 0.5802, Val Acc: 0.7075
Epoch: 19, Train Loss: 0.5172 Val Loss: 0.5973, Val Acc: 0.7015
Epoch: 20, Train Loss: 0.5840 Val Loss: 0.5900, Val Acc: 0.7050
Epoch: 21, Train Loss: 0.4945 Val Loss: 0.5827, Val Acc: 0.7090
Epoch: 22, Train Loss: 0.5311 Val Loss: 0.5844, Val Acc: 0.7065
Epoch: 23, Train Loss: 0.5105 Val Loss: 0.5804, Val Acc: 0.7075
Epoch: 24, Train Loss: 0.5608 Val Loss: 0.5863, Val Acc: 0.6985
Epoch: 25, Train Loss: 0.5879 Val Loss: 0.5878, Val Acc: 0.7020
Epoch: 26, Train Loss: 0.7116 Val Loss: 0.5834, Val Acc: 0.7115
Epoch: 27, Train Loss: 0.6727 Val Loss: 0.5824, Val Acc: 0.7040
Epoch: 28, Train Loss: 0.4441 Val Loss: 0.5821, Val Acc: 0.7080
Epoch: 29, Train Loss: 0.5386 Val Loss: 0.5831, Val Acc: 0.7055
Epoch: 30, Train Loss: 0.5565 Val Loss: 0.5813, Val Acc: 0.7080
```

[ ]:
```
model.eval()
test_correct = 0
for data in test_dataloader:
    data = data.to(device)
    output = model(data)
    pred = output.argmax(dim=1)
    test_correct += int((pred==data.y).sum())

test_acc = test_correct / len(test_dataloader.dataset)
print(f'Test Accuracy: {test_acc:.4f}')
```

```
Test Accuracy: 0.6990
```

[ ]:
```
#test_accuracy = 0.6930, valid_accuracy = 0.7040 for n_neighbors =2
#test_accuracy = 0.6973, valid_accuracy = 0.7025 for n_neighbors =5
#test_accuracy = 0.6990, valid_accuracy = 0.7080 for n_neighbors =10
```

3.4 4) GraphConv

[ ]:
```
class GNN(nn.Module):
    def __init__(self,in_channels ,hidden_dim, output_dim,p=0.3):
        super().__init__()
        self.conv1 = GraphConv(in_channels,hidden_dim)
        self.conv2 = GraphConv(hidden_dim,hidden_dim)
        self.conv3 = GraphConv(hidden_dim,hidden_dim)
        self.fc1 = nn.Linear(hidden_dim,hidden_dim*2)
        self.fc2= nn.Linear(hidden_dim*2,output_dim)
        self.p = p
    def forward(self,data):
        x, edge_index, edge_attr,batch = data.x.float(), data.edge_index, data.
↪edge_attr.float(),data.batch
        x = self.conv1(x,edge_index)
        x = F.relu(x)
        # x = F.dropout(x,p=self.p,training=self.training)
        x = self.conv2(x,edge_index)
        x = F.relu(x)
        # x = F.dropout(x,p=self.p,training=self.training)
        x = self.conv3(x,edge_index)

        x = global_mean_pool(x,batch) #[batch_size,hidden_channels]

        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc1(x)
        x = F.relu(x)
        x = F.dropout(x,p=self.p,training=self.training)
        x = self.fc2(x)
        return x
```

[ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = GNN(in_channels=3,hidden_dim=128,output_dim=2).to(device)
optimizer = torch.optim.Adam(model.parameters(),lr=3e-4)
criterion = nn.CrossEntropyLoss()
```

[ ]:
```
for epoch in range(30):
    model.train()

    for train_data in train_dataloader:
        train_data = train_data.to(device)
        optimizer.zero_grad()
        train_output = model(train_data)
        loss = criterion(train_output, train_data.y)
        loss.backward()
        optimizer.step()

    # Validation
    model.eval()
    val_loss = 0
    val_correct = 0
    for val_data in val_dataloader:
        val_data = val_data.to(device)
        val_output = model(val_data)
        val_loss += criterion(val_output, val_data.y).item()
        pred = val_output.argmax(dim=1)
        # print(len(pred))
        val_correct += int((pred==val_data.y).sum())
        # val_correct += (pred == val_data.y).sum().float()/pred.shape[0]


    val_loss /= len(val_dataloader)
    # print(f'val_correct{val_correct}')
    val_acc = val_correct / len(val_dataloader.dataset)

    print(f'Epoch: {epoch+1}, Train Loss: {loss:.4f} Val Loss: {val_loss:.4f},␣
↪Val Acc: {val_acc:.4f}')
```

```
Epoch: 1, Train Loss: 0.5948 Val Loss: 0.6839, Val Acc: 0.5365
Epoch: 2, Train Loss: 0.5838 Val Loss: 0.6429, Val Acc: 0.6260
Epoch: 3, Train Loss: 0.5251 Val Loss: 0.5970, Val Acc: 0.6875
Epoch: 4, Train Loss: 0.4603 Val Loss: 0.6087, Val Acc: 0.6995
Epoch: 5, Train Loss: 0.5617 Val Loss: 0.5943, Val Acc: 0.6935
Epoch: 6, Train Loss: 0.6682 Val Loss: 0.6160, Val Acc: 0.6790
Epoch: 7, Train Loss: 0.7295 Val Loss: 0.5928, Val Acc: 0.6945
Epoch: 8, Train Loss: 0.6059 Val Loss: 0.6012, Val Acc: 0.7000
Epoch: 9, Train Loss: 0.6289 Val Loss: 0.6364, Val Acc: 0.6265
Epoch: 10, Train Loss: 0.6390 Val Loss: 0.5893, Val Acc: 0.7045
Epoch: 11, Train Loss: 0.6475 Val Loss: 0.5872, Val Acc: 0.7010
Epoch: 12, Train Loss: 0.5155 Val Loss: 0.5896, Val Acc: 0.7020
Epoch: 13, Train Loss: 0.5945 Val Loss: 0.5890, Val Acc: 0.7035
Epoch: 14, Train Loss: 0.5419 Val Loss: 0.5869, Val Acc: 0.7100
Epoch: 15, Train Loss: 0.4081 Val Loss: 0.5814, Val Acc: 0.7075
Epoch: 16, Train Loss: 0.6927 Val Loss: 0.6162, Val Acc: 0.6885
Epoch: 17, Train Loss: 0.6663 Val Loss: 0.5819, Val Acc: 0.7080
Epoch: 18, Train Loss: 0.4436 Val Loss: 0.5809, Val Acc: 0.7095
Epoch: 19, Train Loss: 0.7219 Val Loss: 0.6027, Val Acc: 0.6890
Epoch: 20, Train Loss: 0.4787 Val Loss: 0.5806, Val Acc: 0.7070
Epoch: 21, Train Loss: 0.5317 Val Loss: 0.5764, Val Acc: 0.7115
Epoch: 22, Train Loss: 0.5787 Val Loss: 0.5891, Val Acc: 0.6975
Epoch: 23, Train Loss: 0.5224 Val Loss: 0.5802, Val Acc: 0.7125
Epoch: 24, Train Loss: 0.6407 Val Loss: 0.5765, Val Acc: 0.7210
Epoch: 25, Train Loss: 0.5306 Val Loss: 0.5816, Val Acc: 0.7105
Epoch: 26, Train Loss: 0.3843 Val Loss: 0.5786, Val Acc: 0.7080
Epoch: 27, Train Loss: 0.6315 Val Loss: 0.5770, Val Acc: 0.7180
Epoch: 28, Train Loss: 0.5156 Val Loss: 0.5897, Val Acc: 0.7010
Epoch: 29, Train Loss: 0.4531 Val Loss: 0.5767, Val Acc: 0.7160
Epoch: 30, Train Loss: 0.3989 Val Loss: 0.5736, Val Acc: 0.7170
```

[ ]:
```
model.eval()
test_correct = 0
```

```
for data in test_dataloader:
    data = data.to(device)
    output = model(data)
    pred = output.argmax(dim=1)
    test_correct += int((pred==data.y).sum())

test_acc = test_correct / len(test_dataloader.dataset)
print(f'Test Accuracy: {test_acc:.4f}')
```

```
Test Accuracy: 0.7070
```

[ ]:
```
#test_accuracy = 0.6923, valid_accuracy = 0.7090 for n_neighbors =2
#test_accuracy = 0.6857, valid_accuracy = 0.6930 for n_neighbors =5
#test_accuracy = 0.7070, valid_accuracy = 0.7170 for n_neighbors =10
```

3.5 RESULTS

Model	Test Accuracy	Validation Accuracy
GCN (k=10)	0.6947	0.6950
GCN (k=5)	0.6943	0.6885
GCN (k=2)	0.6887	0.6865
GAT(k=10)	0.6950	0.7060
GAT (k=5)	0.6917	0.6980
GAT (k=2)	0.6870	0.6955
SageConv (k=10)	0.6990	0.7080
SageConv (k=5)	0.6973	0.7025
SageConv (k=2)	0.6930	0.7040
GraphConv (k=10)	0.7070	0.7170
GraphConv (k=5)	0.6857	0.6930
GraphConv (k=2)	0.6923	0.7070

3.6 DISCUSSION

- Accuracy difference between the different vaues of n_neighbors is small, indicating that the various architectures employed are not very sensitive and are robust to different values of n_neighbors
- Brief about architectures used -
    1. GCN operates by aaggregating feature information from neighboring nodes in graph to update central nodes representation. It can only take node features as input.
    2. GAT introduces attention mechanisms to weigh importance of neighboring nodes when aggregating information. It can take both node features and edge features.
    3. SageConv operates by sampling and aggregating features from neigboring nodes. It incorporates pooling operations to aggregate information from neighboring nodes. It can only take node features as input.

4. GraphConv aggregates information from neighboring nodes using weighted combination of node features.It can only take node features as input.

- All of them achieve an accuracy in range of 68-70%, an method to improve the accuracy could be to either deepen the current neural network architectures or apply networks that could model longer range dependencies and capture more complex patterns like Graph Transformer Networks or Graph Isomorphism networks.

3.7 REFERNCES -

1. https://arxiv.org/pdf/2104.01725.pdf
2. https://ml4physicalsciences.github.io/2020/files/NeurIPS_ML4PS_2020_138.pdf
3. https://pytorch-geometric.readthedocs.io/en/latest/get_started/colabs.html