AI VIET NAM
@aivietnam.edu.vn

Graph Neural Network – Exercise

TA Minh-Duc Bui

Outline

1. Objective
2. From MLP to GNN
3. Coding
4. Experiment on Caltech-101 and PACS Datasets

Outline

1. Objective
2. From MLP to GNN
3. Coding
4. Experiment on Caltech-101 and PACS Datasets

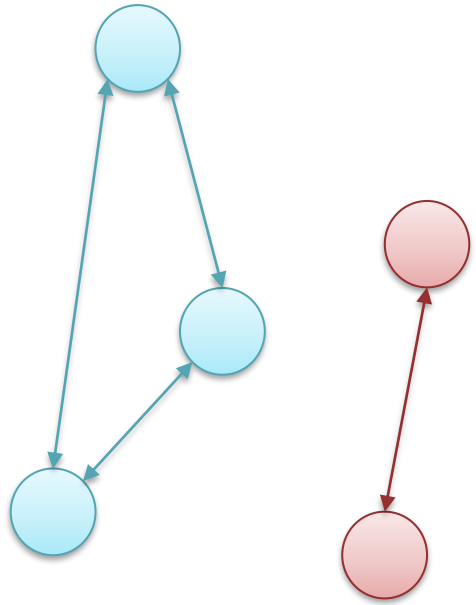
AI VIET NAM
@aivietnam.edu.vn

Objective

To utilize GNNs (Graph Neural Networks) for any problem, three sequential questions need to be addressed:

- What is a node?
- What is an edge?
- Can we incorporate an auxiliary loss function?

How does GNN operate?

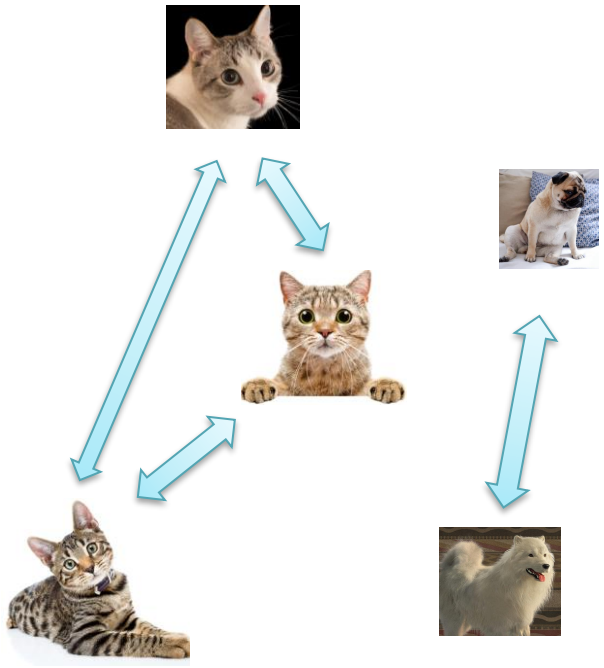


Objective

To utilize GNNs (Graph Neural Networks) for any problem, three sequential questions need to be addressed:

- What is a node?
- What is an edge?
- Can we incorporate an auxiliary loss function?

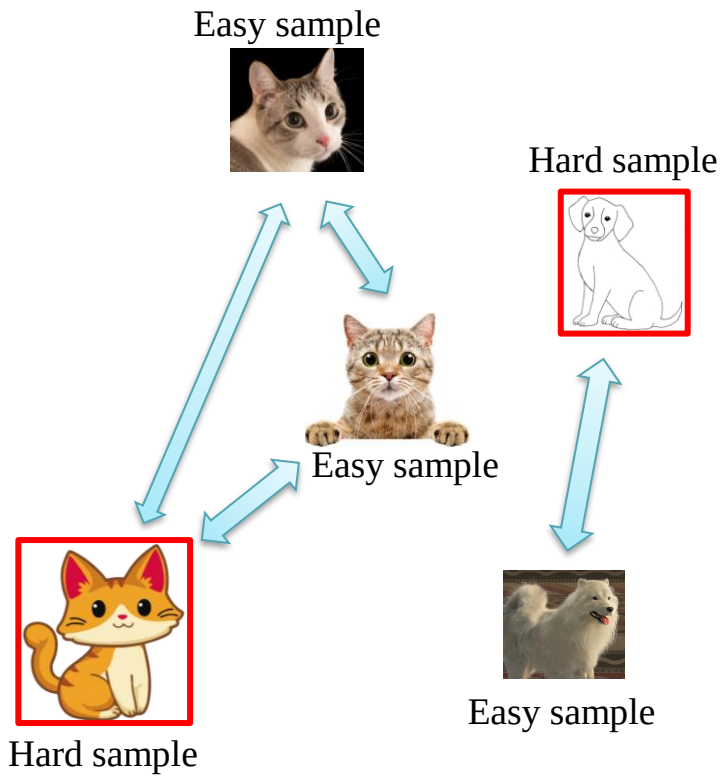
Node: image
Edge: images of the same class
Auxiliary loss: yes (because it's the groundtruth)



Objective

How does GNN operate?

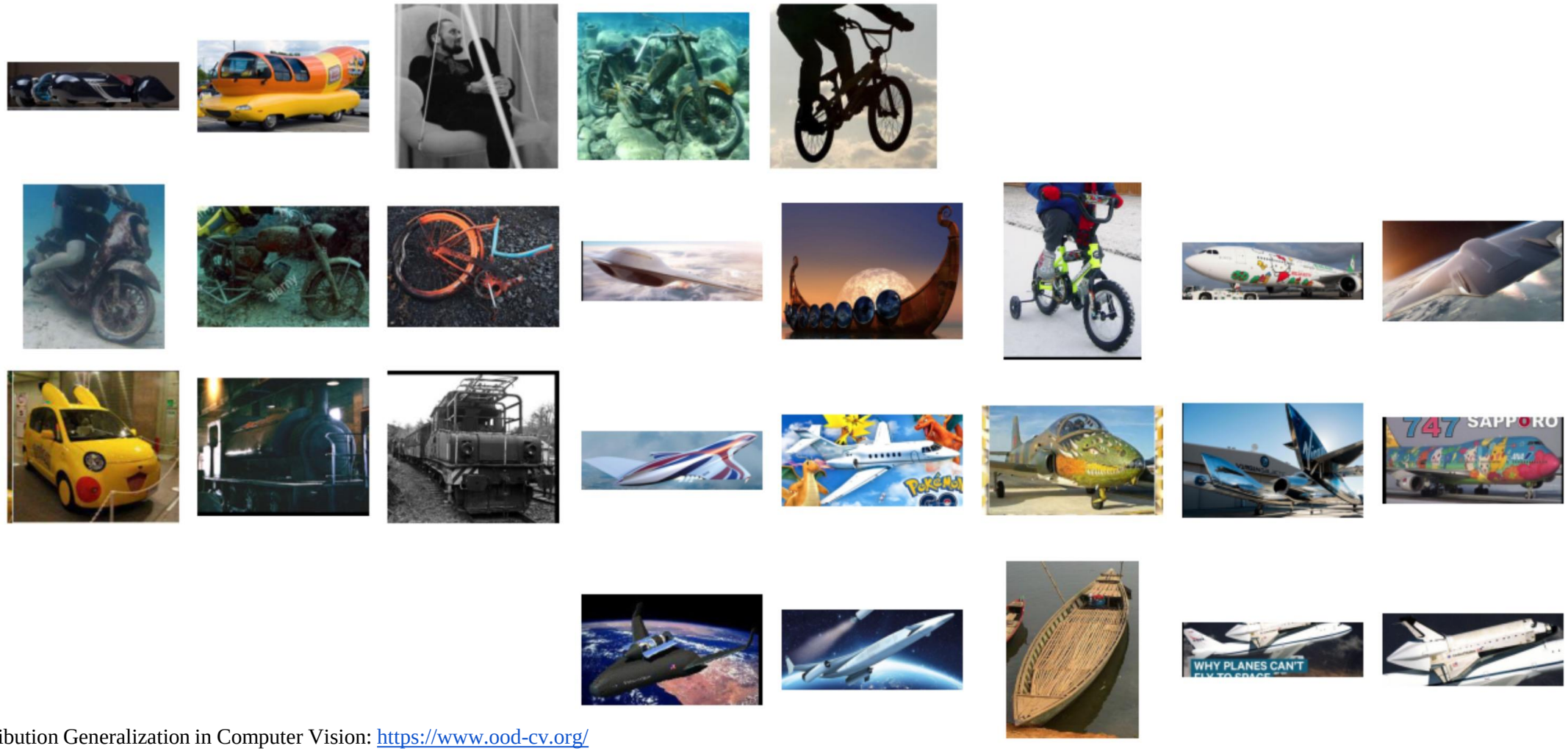
- There are hard and easy samples for each class.
- Leverage easy samples to “provide” information for hard samples.



Objective

Examples of hard samples

Out-of-Distribution



Out Of Distribution Generalization in Computer Vision: https://www.ood-cv.org/

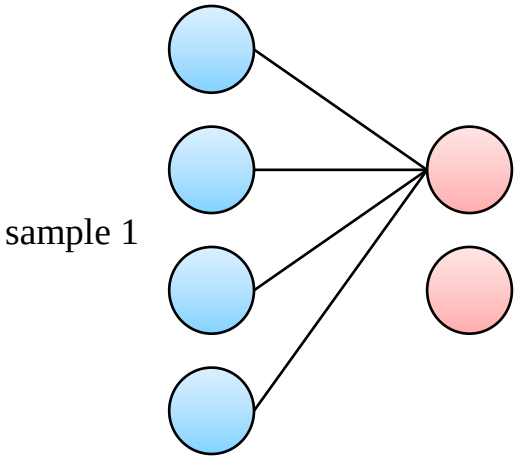
Outline

1. Objective
2. From MLP to GNN
3. Coding
4. Experiment on Caltech-101 and PACS Datasets

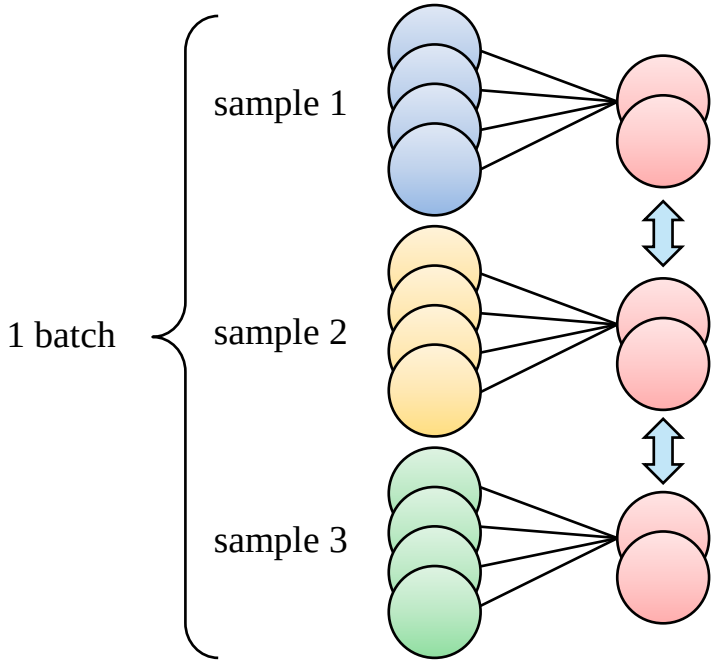
From MLP to GNN

MLP

Sample-view

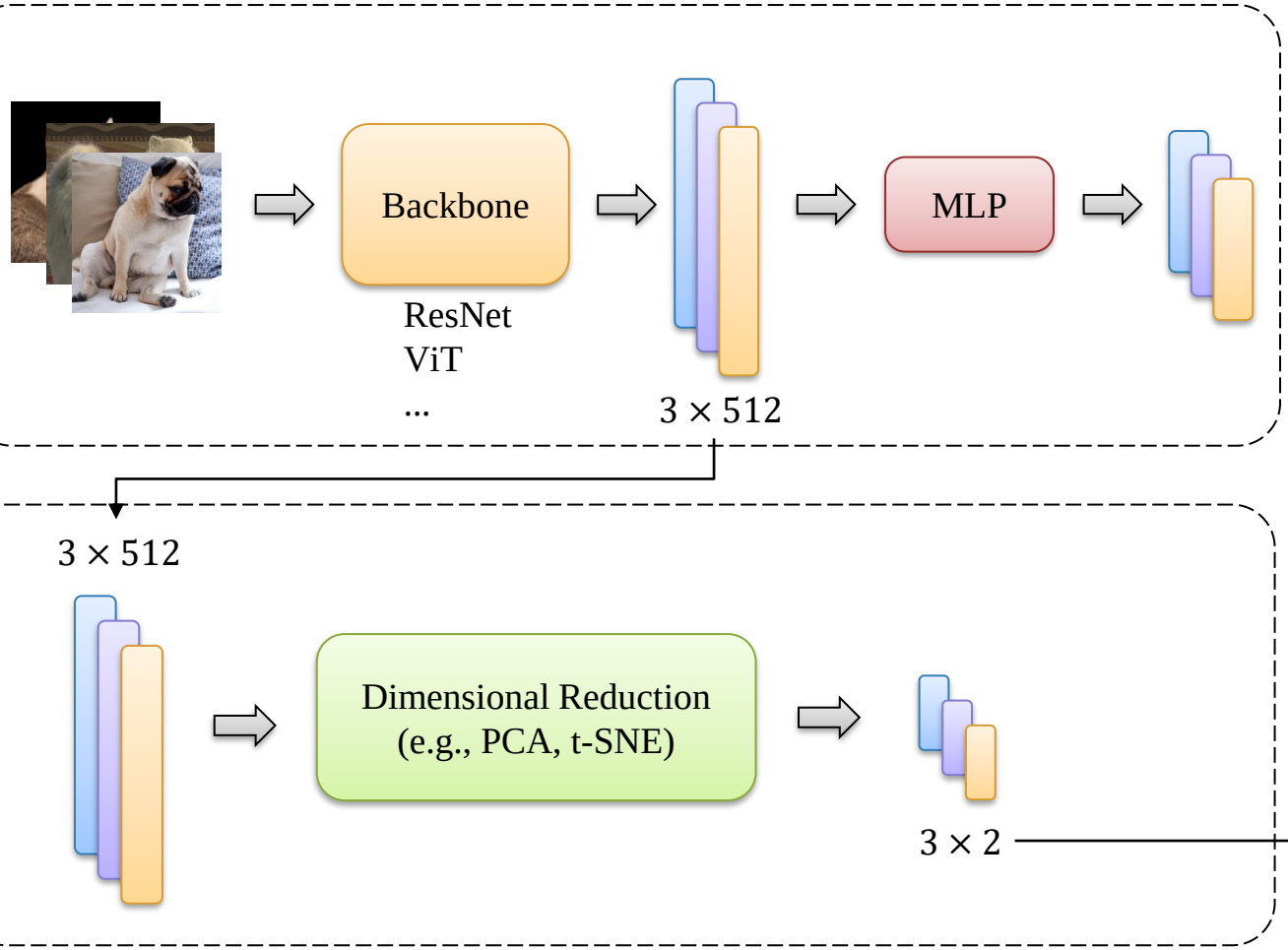


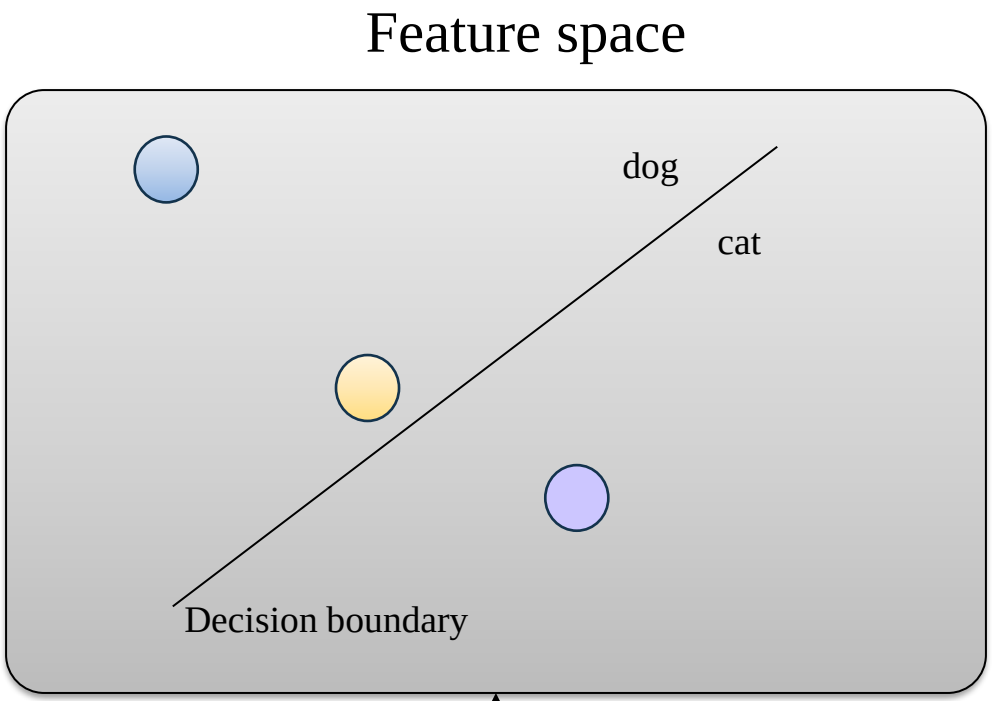
Batch-view



From MLP to GNN

Feature space

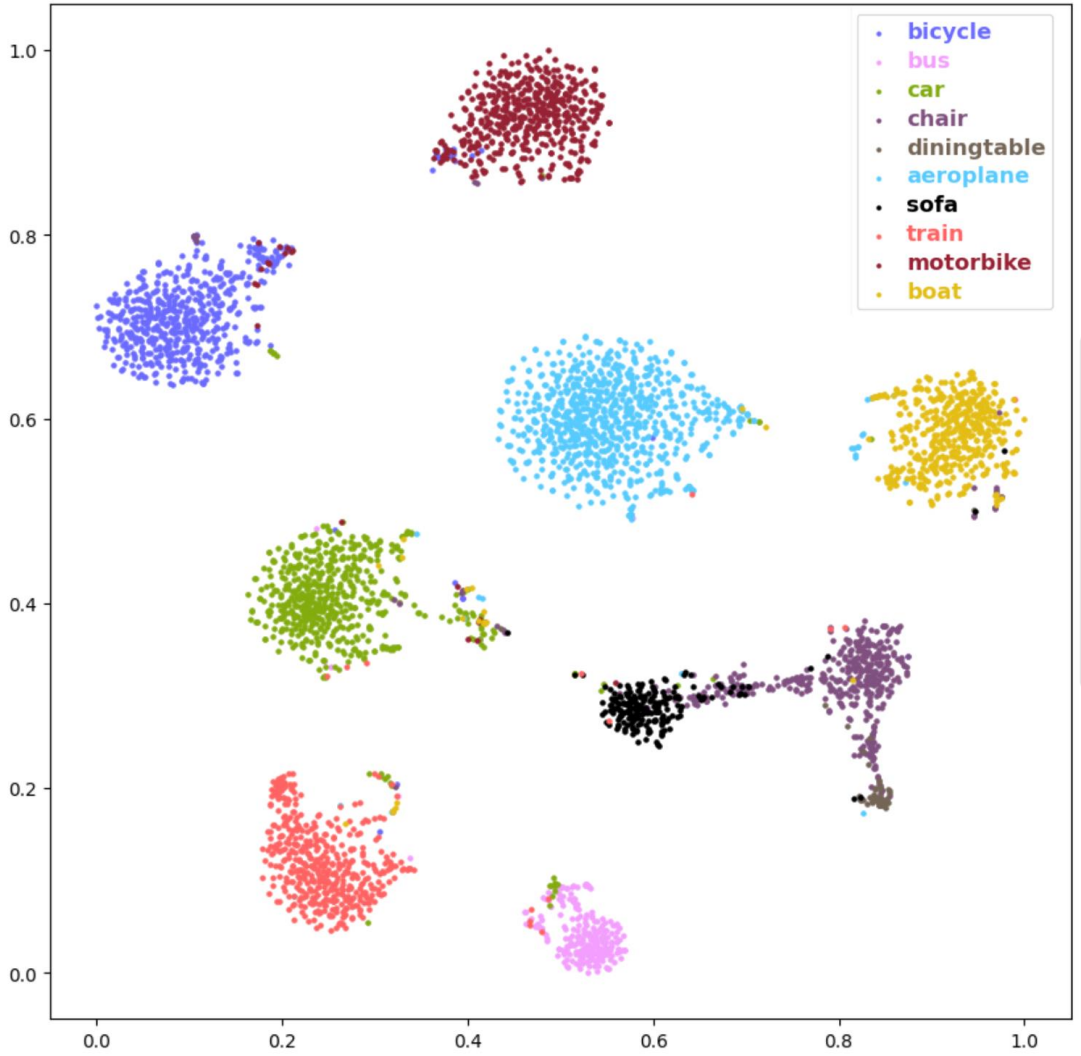




AI VIET NAM
@aivietnam.edu.vn

From MLP to GNN

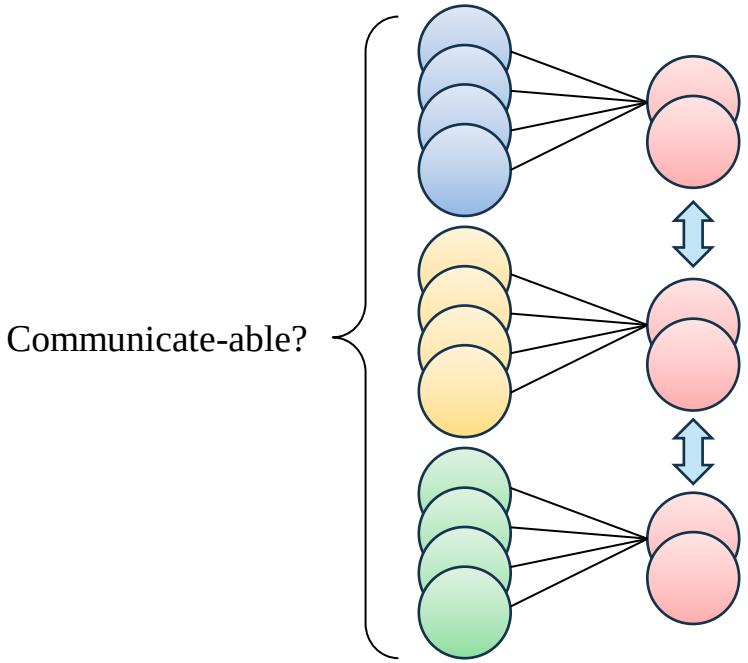
Feature space

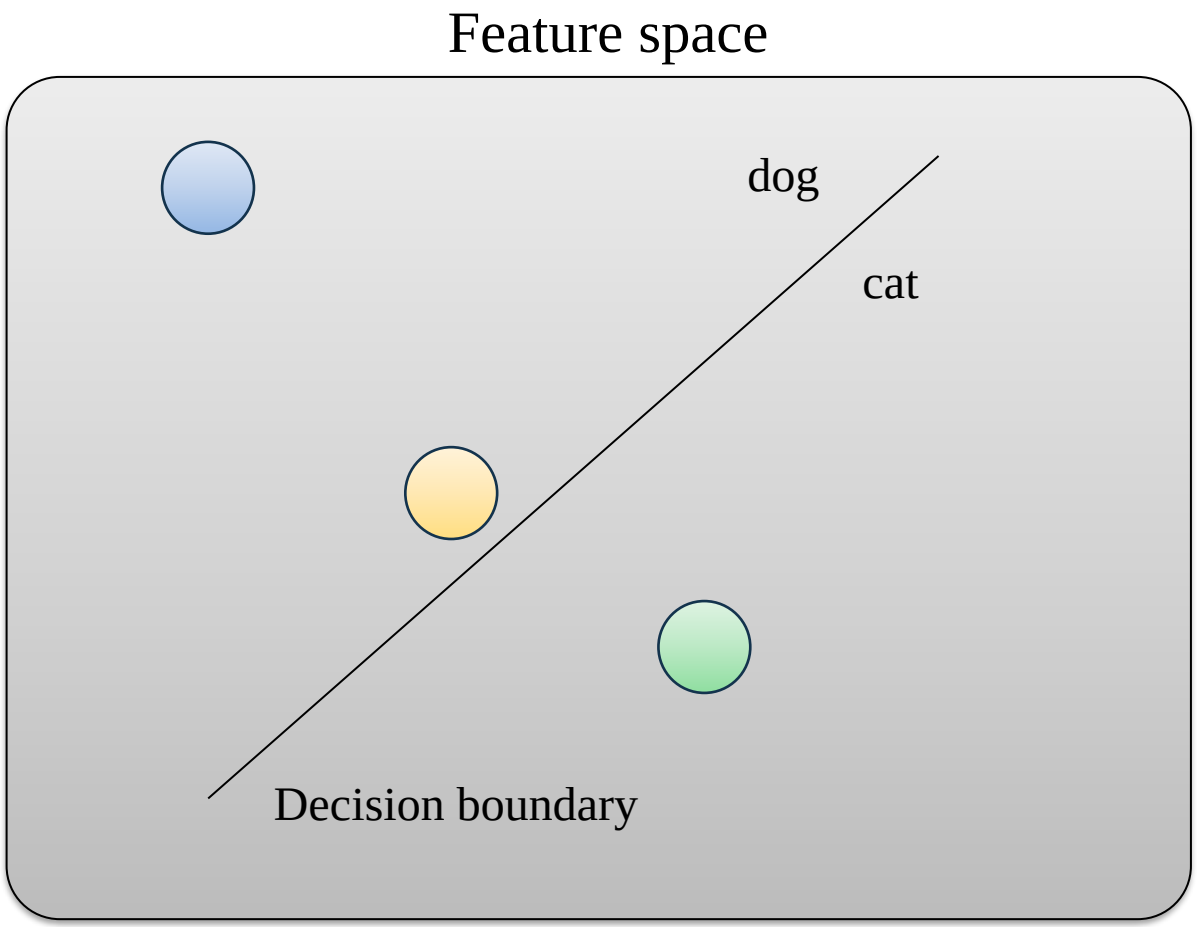


t-SNE Visualization

From MLP to GNN

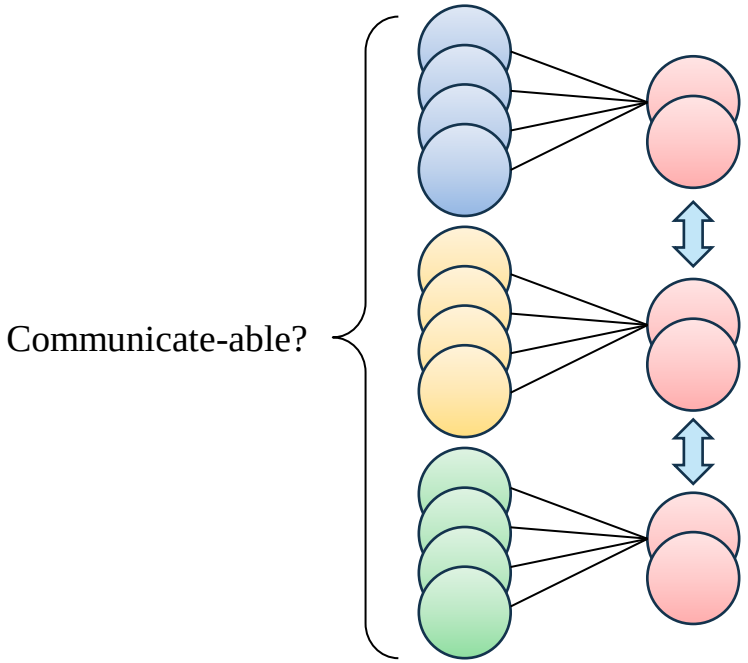
MLP

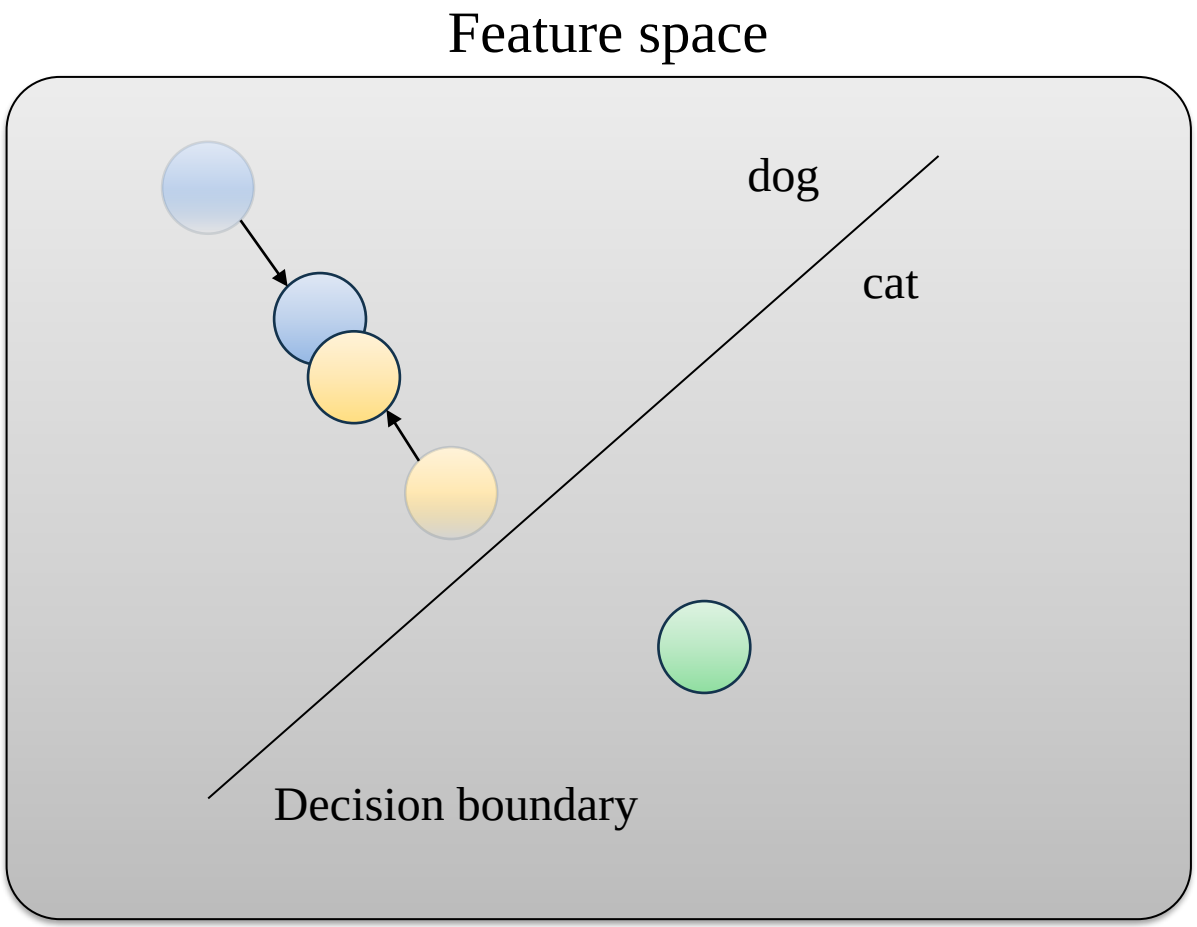




From MLP to GNN

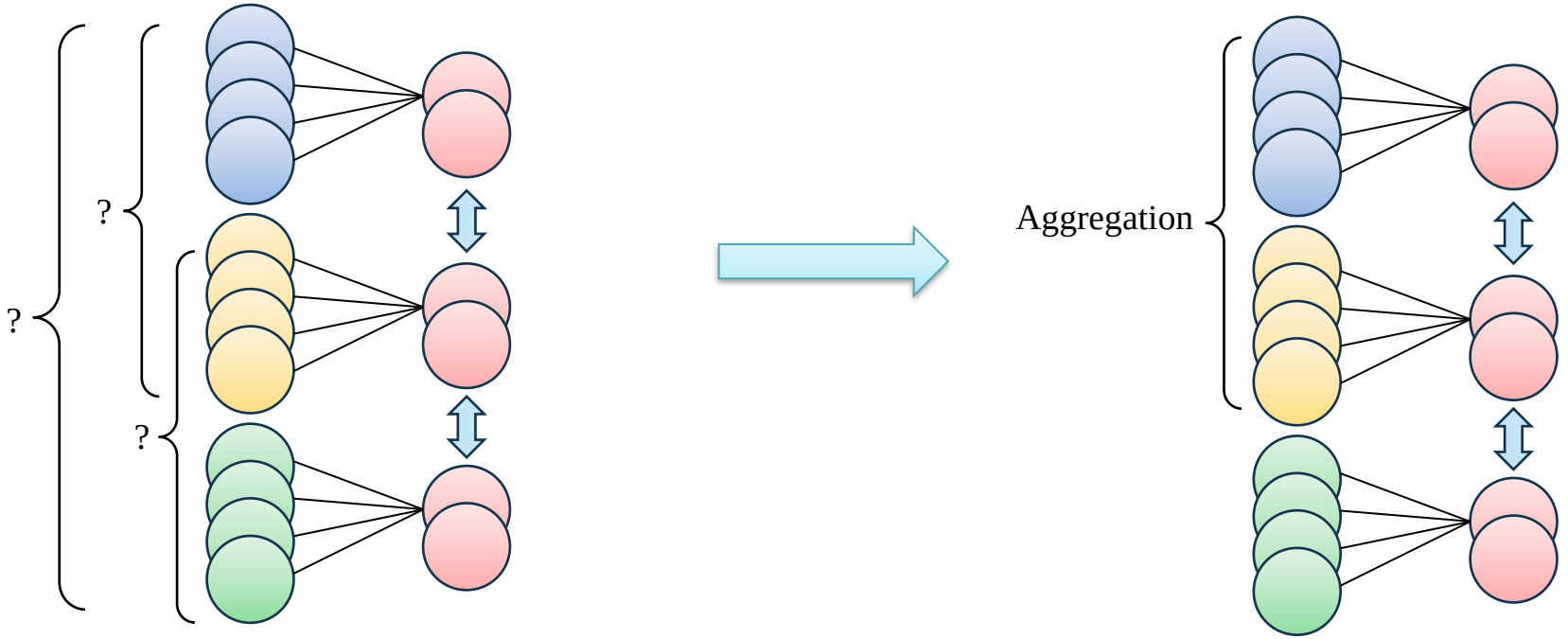
MLP





From MLP to GNN

How to communicate?



Edge Network

Node Network

GNN = MLP + Message Passing (MP)

GNN

(MP): $\tilde{\mathbf{h}}_u^{(l-1)} = \sum_{v \in \mathcal{N}_u \cup \{u\}} a_{u,v} \cdot \mathbf{h}_v^{(l-1)}$, (FF): $\mathbf{h}_u^{(l)} = \psi^{(l)}\left(\tilde{\mathbf{h}}_u^{(l-1)}\right)$ (1)

S.t.
$\psi^{(l)}$ denotes a feature transformation mapping at the l-th layer → nn.Linear()
$a_{\mathcal{G}}(u, v)$ is the affinity function
$\mathbf{h}_u^{(0)} = \boldsymbol{x}_u$

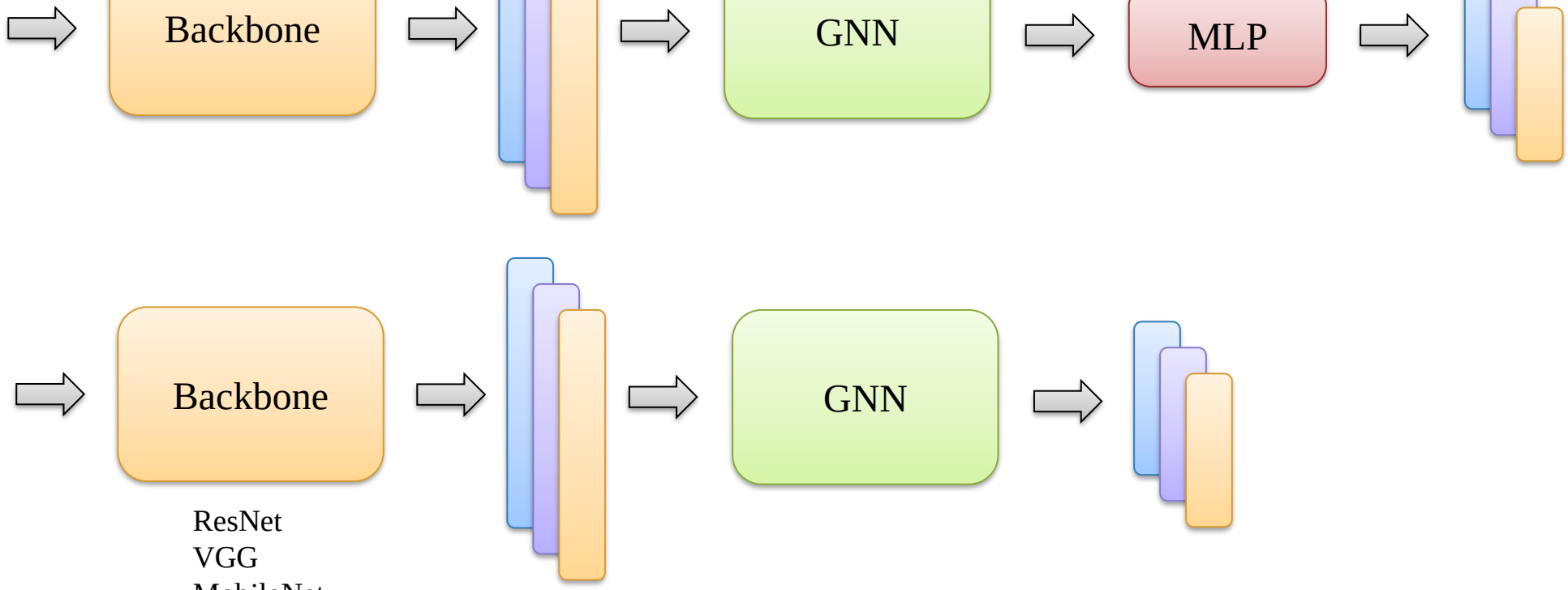
MLP

GNN models in forms of **Eq. 1** degrade to an **MLP** with a series of FF layers after removing all the MP operations:

$$\hat{y}_u = \psi^{(L)}(\cdots(\psi^{(1)}(\mathbf{x}_u)) = \psi(\mathbf{x}_u). \quad (2)$$

```
# GNN
fc1 = nn.Linear()
mp1 = MP()
fc2 = nn.Linear()
mp2 = MP()

# Forward
out1 = fc1(mp1(x))
out2 = fc2(mp2(out1))

-----------------

# MLP
fc1 = nn.Linear()
fc2 = nn.Linear()

# Forward
out1 = fc1(x)
out2 = fc2(out1)
```

From MLP to GNN

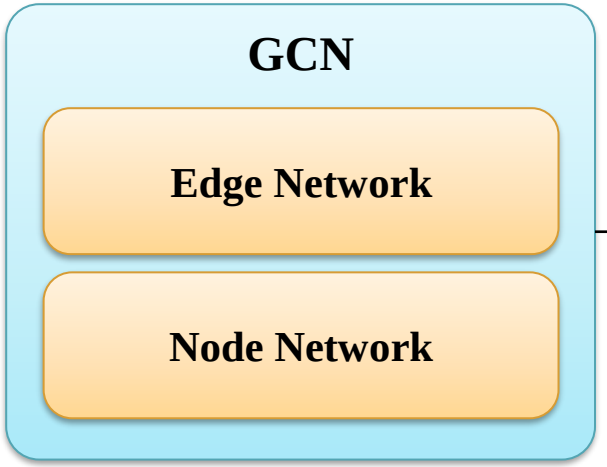


ResNet
VGG
MobileNet
ViT
Swin
...

Outline

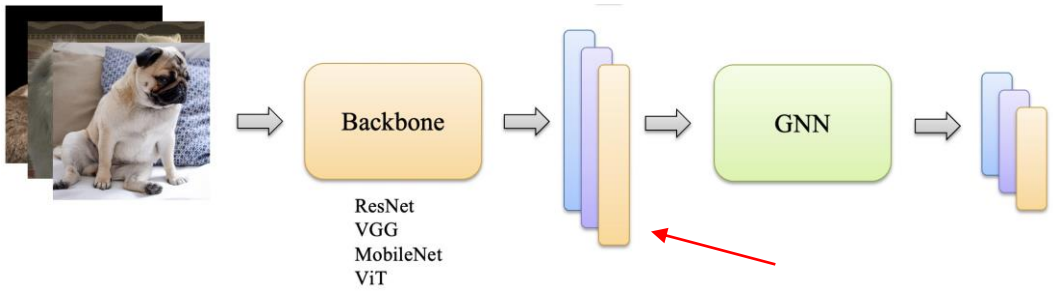
1. Objective
2. From MLP to GNN
3. **Coding**
4. Experiment on Caltech-101 and PACS Datasets

GCN



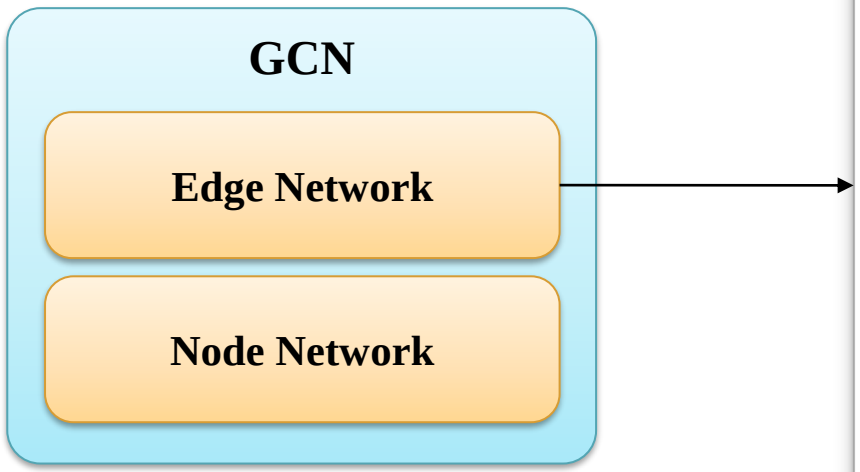
Input = init_node_feat (bs, hidden_dim)
Output = logits_gnn (bs, hidden_dim)

edge_feat (bs, bs): correlation matrix **after** normalizing
edge_sim (bs, bs): correlation matrix **before** normalizing (for loss function)




```python
class GCN(nn.Module):
    def __init__(self, in_features, edge_features, out_feature, device, ratio=(1,)):
        super(GCN, self).__init__()

        self.edge_net = EdgeNet(
            in_features=in_features,
            num_features=edge_features,
            device=device,
            ratio=ratio,
        )
        # set edge to node
        self.node_net = NodeNet(
            in_features=in_features,
            num_features=out_feature,
            device=device,
            ratio=ratio,
        )
        # mask value for no-gradient edges
        self.mask_val = -1

    def label2edge(self, targets):
        """convert node labels to affinity mask for backprop"""
        num_sample = targets.size()[1]
        label_i = targets.unsqueeze(-1).repeat(1, 1, num_sample)
        label_j = label_i.transpose(1, 2)
        edge = torch.eq(label_i, label_j).float()
        target_edge_mask = (
            torch.eq(label_i, self.mask_val) + torch.eq(label_j, self.mask_val)
        ).type(torch.bool)
        source_edge_mask = ~target_edge_mask
        init_edge = edge * source_edge_mask.float()
        return init_edge[0], source_edge_mask

    def forward(self, init_node_feat):
        #  compute normalized and not normalized affinity matrix
        edge_feat, edge_sim = self.edge_net(init_node_feat)
        # compute node features and class logits
        logits_gnn = self.node_net(init_node_feat, edge_feat)
        return logits_gnn, edge_sim
```

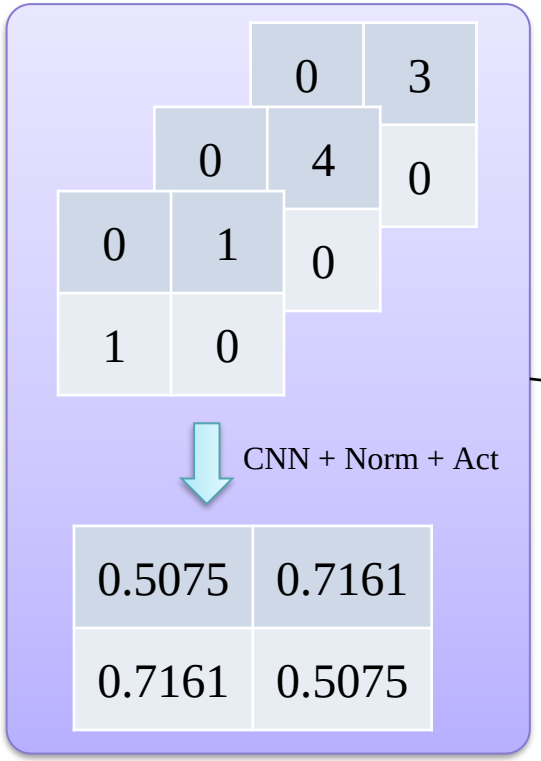
Edge Network




```python
class EdgeNet(nn.Module):
    def __init__(self, in_features, num_features, device, ratio=(1,)):
        super(EdgeNet, self).__init__()
        num_features_list = [num_features * r for r in ratio]
        self.device = device
        # define layers
        layer_list = OrderedDict()
        for l in range(len(num_features_list)):
            layer_list["conv{}".format(l)] = nn.Conv2d(
                in_channels=num_features_list[l - 1] if l > 0 else in_features,
                out_channels=num_features_list[l],
                kernel_size=1,
                bias=False,
            )
            layer_list["norm{}".format(l)] = nn.BatchNorm2d(
                num_features=num_features_list[l]
            )
            layer_list["relu{}".format(l)] = nn.LeakyReLU()
        # add final similarity kernel
        layer_list["conv_out"] = nn.Conv2d(
            in_channels=num_features_list[-1], out_channels=1, kernel_size=1
        )
        self.sim_network = nn.Sequential(layer_list).to(device)

    def forward(self, node_feat):
        node_feat = node_feat.unsqueeze(dim=0)  # (1, bs, dim)
        num_tasks = node_feat.size(0)  # 1
        num_data = node_feat.size(1)  # bs
        x_i = node_feat.unsqueeze(2)  # (1, bs, 1, dim)
        x_j = torch.transpose(x_i, 1, 2)  # (1, 1, bs, dim)
        x_ij = torch.abs(x_i - x_j)  # (1, bs, bs, dim)
        x_ij = torch.transpose(x_ij, 1, 3)  # (1, dim, bs, bs)
        # compute similarity/dissimilarity (batch_size x feat_size x num_samples x num_samples)
        sim_val = (
            torch.sigmoid(self.sim_network(x_ij)).squeeze(1).squeeze(0).to(self.device)
        )  # (bs, bs)
        # normalize affinity matrix
        force_edge_feat = (
            torch.eye(num_data).unsqueeze(0).repeat(num_tasks, 1, 1).to(self.device)
        )  # (1, bs, bs)
        edge_feat = sim_val + force_edge_feat  # (bs, bs)
        edge_feat = edge_feat + 1e-6  # add small value to avoid nan
        edge_feat = edge_feat / torch.sum(edge_feat, dim=1).unsqueeze(1)  # normalize
        return edge_feat, sim_val  # (bs, bs), (bs, bs)
```

Edge Network

	[1 2 3]	[2 6 6]
[1 2 3]	[0 0 0]	[1 4 3]
[2 6 6]	[1 4 3]	[0 0 0]

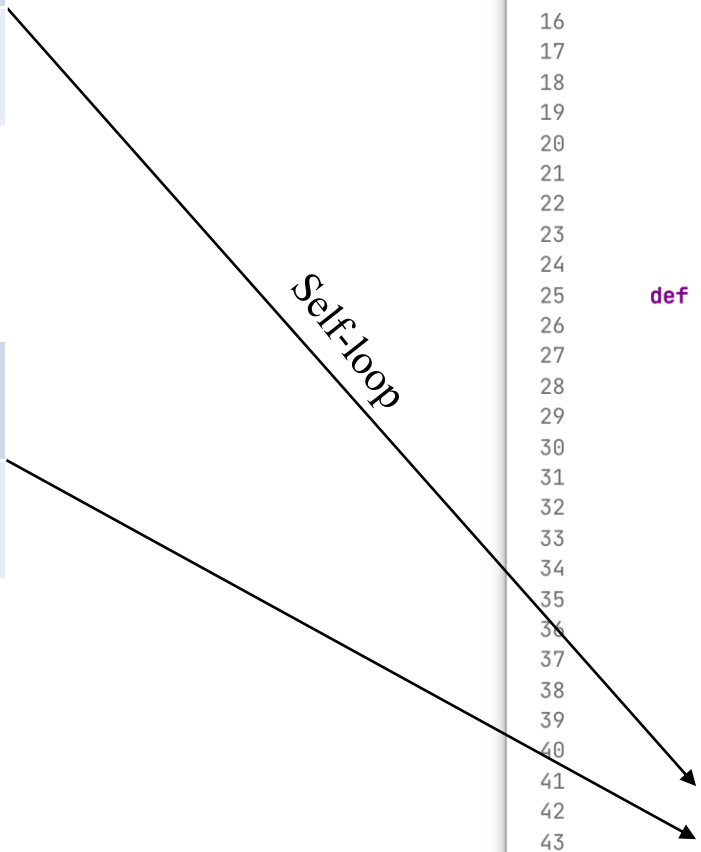



```python
class EdgeNet(nn.Module):
    def __init__(self, in_features, num_features, device, ratio=(1,)):
        super(EdgeNet, self).__init__()
        num_features_list = [num_features * r for r in ratio]
        self.device = device
        # define layers
        layer_list = OrderedDict()
        for l in range(len(num_features_list)):
            layer_list["conv{}".format(l)] = nn.Conv2d(
                in_channels=num_features_list[l - 1] if l > 0 else in_features,
                out_channels=num_features_list[l],
                kernel_size=1,
                bias=False,
            )
            layer_list["norm{}".format(l)] = nn.BatchNorm2d(
                num_features=num_features_list[l]
            )
            layer_list["relu{}".format(l)] = nn.LeakyReLU()
        # add final similarity kernel
        layer_list["conv_out"] = nn.Conv2d(
            in_channels=num_features_list[-1], out_channels=1, kernel_size=1
        )
        self.sim_network = nn.Sequential(layer_list).to(device)

    def forward(self, node_feat):
        node_feat = node_feat.unsqueeze(dim=0)  # (1, bs, dim)
        num_tasks = node_feat.size(0)  # 1
        num_data = node_feat.size(1)  # bs
        x_i = node_feat.unsqueeze(2)  # (1, bs, 1, dim)
        x_j = torch.transpose(x_i, 1, 2)  # (1, 1, bs, dim)
        x_ij = torch.abs(x_i - x_j)  # (1, bs, bs, dim)
        x_ij = torch.transpose(x_ij, 1, 3)  # (1, dim, bs, bs)
        # compute similarity/dissimilarity (batch_size x feat_size x num_samples x num_samples)
        sim_val = (
            torch.sigmoid(self.sim_network(x_ij)).squeeze(1).squeeze(0).to(self.device)
        )  # (bs, bs)
        # normalize affinity matrix
        force_edge_feat = (
            torch.eye(num_data).unsqueeze(0).repeat(num_tasks, 1, 1).to(self.device)
        )  # (1, bs, bs)
        edge_feat = sim_val + force_edge_feat  # (bs, bs)
        edge_feat = edge_feat + 1e-6  # add small value to avoid nan
        edge_feat = edge_feat / torch.sum(edge_feat, dim=1).unsqueeze(1)  # normalize
        return edge_feat, sim_val  # (bs, bs), (bs, bs)
```

Edge Network

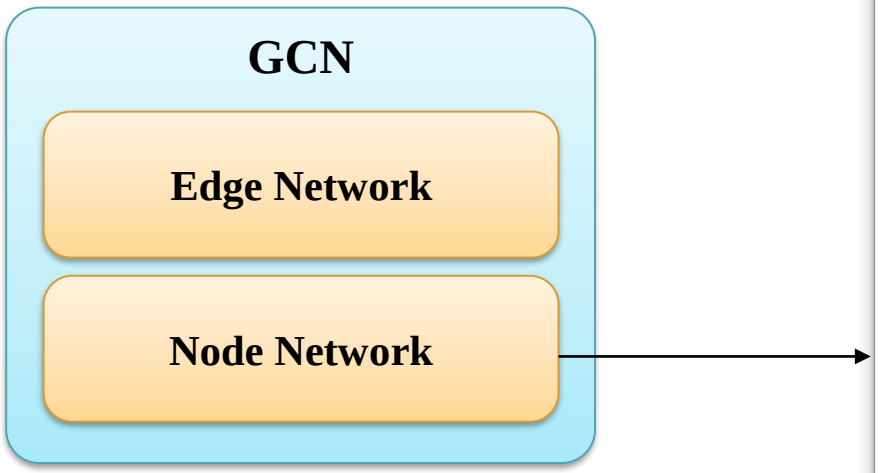
1.5075	0.7161
0.7161	1.5075

0.8487	0.1513
0.1513	0.8487




```python
class EdgeNet(nn.Module):
    def __init__(self, in_features, num_features, device, ratio=(1,)):
        super(EdgeNet, self).__init__()
        num_features_list = [num_features * r for r in ratio]
        self.device = device
        # define layers
        layer_list = OrderedDict()
        for l in range(len(num_features_list)):
            layer_list["conv{}".format(l)] = nn.Conv2d(
                in_channels=num_features_list[l - 1] if l > 0 else in_features,
                out_channels=num_features_list[l],
                kernel_size=1,
                bias=False,
            )
            layer_list["norm{}".format(l)] = nn.BatchNorm2d(
                num_features=num_features_list[l]
            )
            layer_list["relu{}".format(l)] = nn.LeakyReLU()
        # add final similarity kernel
        layer_list["conv_out"] = nn.Conv2d(
            in_channels=num_features_list[-1], out_channels=1, kernel_size=1
        )
        self.sim_network = nn.Sequential(layer_list).to(device)

    def forward(self, node_feat):
        node_feat = node_feat.unsqueeze(dim=0)  # (1, bs, dim)
        num_tasks = node_feat.size(0)  # 1
        num_data = node_feat.size(1)  # bs
        x_i = node_feat.unsqueeze(2)  # (1, bs, 1, dim)
        x_j = torch.transpose(x_i, 1, 2)  # (1, 1, bs, dim)
        x_ij = torch.abs(x_i - x_j)  # (1, bs, bs, dim)
        x_ij = torch.transpose(x_ij, 1, 3)  # (1, dim, bs, bs)
        # compute similarity/dissimilarity (batch_size x feat_size x num_samples x num_samples)
        sim_val = (
            torch.sigmoid(self.sim_network(x_ij)).squeeze(1).squeeze(0).to(self.device)
        )  # (bs, bs)
        # normalize affinity matrix
        force_edge_feat = (
            torch.eye(num_data).unsqueeze(0).repeat(num_tasks, 1, 1).to(self.device)
        )  # (1, bs, bs)
        edge_feat = sim_val + force_edge_feat  # (bs, bs)
        edge_feat = edge_feat + 1e-6  # add small value to avoid nan
        edge_feat = edge_feat / torch.sum(edge_feat, dim=1).unsqueeze(1)  # normalize
        return edge_feat, sim_val  # (bs, bs), (bs, bs)
```

Node Network




```python
class NodeNet(nn.Module):
    def __init__(self, in_features, num_features, device, ratio=(1,)):
        super(NodeNet, self).__init__()
        num_features_list = [num_features * r for r in ratio]
        self.device = device
        # define layers
        layer_list = OrderedDict()
        for l in range(len(num_features_list)):
            layer_list["conv{}".format(l)] = nn.Conv2d(
                in_channels=num_features_list[l - 1] if l > 0 else in_features * 2,
                out_channels=num_features_list[l],
                kernel_size=1,
                bias=False,
            )
            layer_list["norm{}".format(l)] = nn.BatchNorm2d(
                num_features=num_features_list[l]
            )
            if l < (len(num_features_list) - 1):
                layer_list["relu{}".format(l)] = nn.LeakyReLU()
        self.network = nn.Sequential(layer_list).to(device)

    def forward(self, node_feat, edge_feat):
        """node_feat: (bs, dim), edge_feat: (bs, bs)"""
        node_feat = node_feat.unsqueeze(dim=0)  # (1, bs, dim)
        num_tasks = node_feat.size(0)  # 1
        num_data = node_feat.size(1)  # bs
        # get eye matrix (batch_size x node_size x node_size) only use inter dist.
        diag_mask = 1.0 - torch.eye(num_data).unsqueeze(0).repeat(num_tasks, 1, 1).to(
            self.device
        )  # (1, bs, bs)
        # set diagonal as zero and normalize
        edge_feat = F.normalize(edge_feat * diag_mask, p=1, dim=-1)  # (bs, bs)
        # compute attention and aggregate
        aggr_feat = torch.bmm(edge_feat.squeeze(1), node_feat)  # (bs, dim)
        node_feat = torch.cat([node_feat, aggr_feat], -1).transpose(
            1, 2
        )  # (1, 2*dim, bs)
        # non-linear transform
        node_feat = self.network(node_feat.unsqueeze(-1)).transpose(
            1, 2
        )  # (1, bs, dim)
        node_feat = node_feat.squeeze(-1).squeeze(0)  # (bs, dim)
        return node_feat
```

AI VIET NAM
@aivietnam.edu.vn

Add GCN into Existing Code

```python
from torchvision.models import mobilenet_v3_small

class Model(nn.Module):
    def __init__(self, num_classes=2):
        super(Model, self).__init__()
        self.backbone = mobilenet_v3_small(pretrained=True)
        self.backbone.classifier = nn.Sequential()

        self.head = nn.Linear(576, num_classes)

    def forward(self, x):
        x = self.backbone(x)
        x = self.head(x)
        return x
```

```python
from torchvision.models import mobilenet_v3_small

class Model(nn.Module):
    def __init__(self, num_classes=7):
        super(Model, self).__init__()
        self.backbone = mobilenet_v3_small(pretrained=True)
        self.backbone.classifier = nn.Sequential()

        self.gcn = GCN(
            in_features=576,
            edge_features=576,
            out_feature=num_classes,
            device="cuda",
            ratio=(1,),
        )

    def forward(self, x):
        x = self.backbone(x)
        x, edge_sim = self.gcn(x)
        return x, edge_sim
```

AI VIET NAM
@aivietnam.edu.vn

Add GCN into Existing Code

```python
criterion = nn.CrossEntropyLoss()

for i, (inputs, labels) in enumerate(trainloader):
    # Move inputs and labels to the device
    inputs, labels = inputs.to(device), labels.to(device)

    # Zero the parameter gradients
    optimizer.zero_grad()

    # Forward pass
    outputs = model(inputs)

    # loss
    loss = criterion(outputs, labels)

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
```

```python
criterion = nn.CrossEntropyLoss()
criterion_edge = nn.BCELoss()

for i, (inputs, labels) in enumerate(trainloader):
    # Move inputs and labels to the device
    inputs, labels = inputs.to(device), labels.to(device)

    # Zero the parameter gradients
    optimizer.zero_grad()

    # Forward pass
    outputs, edge_sim = model(inputs)

    ######### Cls loss
    loss_cls = criterion(outputs, labels)

    ######### Edge loss
    edge_gt, edge_mask = model.gcn.label2edge(labels.unsqueeze(dim=0))
    loss_edge = criterion_edge(
        edge_sim.masked_select(edge_mask), edge_gt.masked_select(edge_mask)
    )

    ######### Total loss
    loss = 0.3 * loss_cls + loss_edge

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
```

Create Label for Edge Network

```python
criterion = nn.CrossEntropyLoss()
criterion_edge = nn.BCELoss()

for i, (inputs, labels) in enumerate(trainloader):
    # Move inputs and labels to the device
    inputs, labels = inputs.to(device), labels.to(device)

    # Zero the parameter gradients
    optimizer.zero_grad()

    # Forward pass
    outputs, edge_sim = model(inputs)

    ######### Cls loss
    loss_cls = criterion(outputs, labels)

    ######### Edge loss
    edge_gt, edge_mask = model.gcn.label2edge(labels.unsqueeze(dim=0))
    loss_edge = criterion_edge(
        edge_sim.masked_select(edge_mask), edge_gt.masked_select(edge_mask)
    )

    ######### Total loss
    loss = 0.3 * loss_cls + loss_edge

    # Backward pass and optimization
    loss.backward()
    optimizer.step()
```

labels

1	5	3	1

edge_gt

1	0	0	1
0	1	0	0
0	0	1	0
1	0	0	1

AI VIET NAM

@aivietnam.edu.vn

QUIZ! TIME

AI VIET NAM
@aivietnam.edu.vn

Quiz

1. Đâu là lưu ý khi áp dụng GNN vào các bài toán bất kì?
A. Label là gì?
B. Độ lớn dataset
C. Độ phức tạp của model
D. Edge là gì?

2. Đâu là lưu ý khi áp dụng GNN vào các bài toán bất kì?
A. Label là gì?
B. Node là gì?
C. Số lượng param
D. Loss là gì?

3. Đâu là lưu ý khi áp dụng GNN vào các bài toán bất kì?
A. Có cần hàm loss phụ trợ hay không?
B. Model có tính toán song song hay không?
C. Loss là gì?
D. Loại hàm loss của model hiện tại.

4. Sau khi chắc chắn GNN có thể sử dụng, điều gì ta cần cân nhắc tiếp theo là phù hợp nhất?
A. Không cần gì
B. Độ lớn của model sau khi thêm GNN
C. Hiểu rõ cách GNN tổng hợp thông tin
D. Tốc độ của model sau khi thêm GNN

5. Nếu input của GNN có shape [batch_size, dim_1] và output có shape [batch_size, dim_2] thì nhận định nào sau đây là **SAI**:
A. dim_2 có thể bằng dim_1* 2
B. dim_2 có thể bằng số lượng class
C. dim_1 và dim_2 không bắt buộc bằng nhau
D. dim_1 và dim_ 2 phải bằng nhau

Outline

1. Objective
2. From MLP to GNN
3. Coding
4. **Experiment on Caltech-101 and PACS Datasets**

The Caltech-101 Dataset

Caltech-101 consists of pictures of objects belonging to 101 classes.

Each image is labeled with a single object.

Each class contains roughly 40 to 800 images, totaling around 9k images.

Images are of variable sizes, with typical edge lengths of 200-300 pixels.

The Caltech101 Dataset

```python
batch_size = 256
train_transform = transforms.Compose(
    [
        transforms.Resize((224, 224)),
        transforms.Grayscale(num_output_channels=3),
        transforms.ToTensor(),
        transforms.Normalize([0.4914, 0.4822, 0.4465],
                             [0.2470, 0.2435, 0.2616]),
    ]
)


dataset = Caltech101(root="./data", download=True,
                     transform=train_transform)

# # split dataset
n_train = int(len(dataset) * 0.8)
n_val = len(dataset) - n_train

train_dataset, val_dataset = random_split(dataset, [n_train, n_val])

trainloader = DataLoader(train_dataset,
                         batch_size=batch_size,
                         shuffle=True)
testloader = DataLoader(val_dataset,
                        batch_size=batch_size,
                        shuffle=False)
```

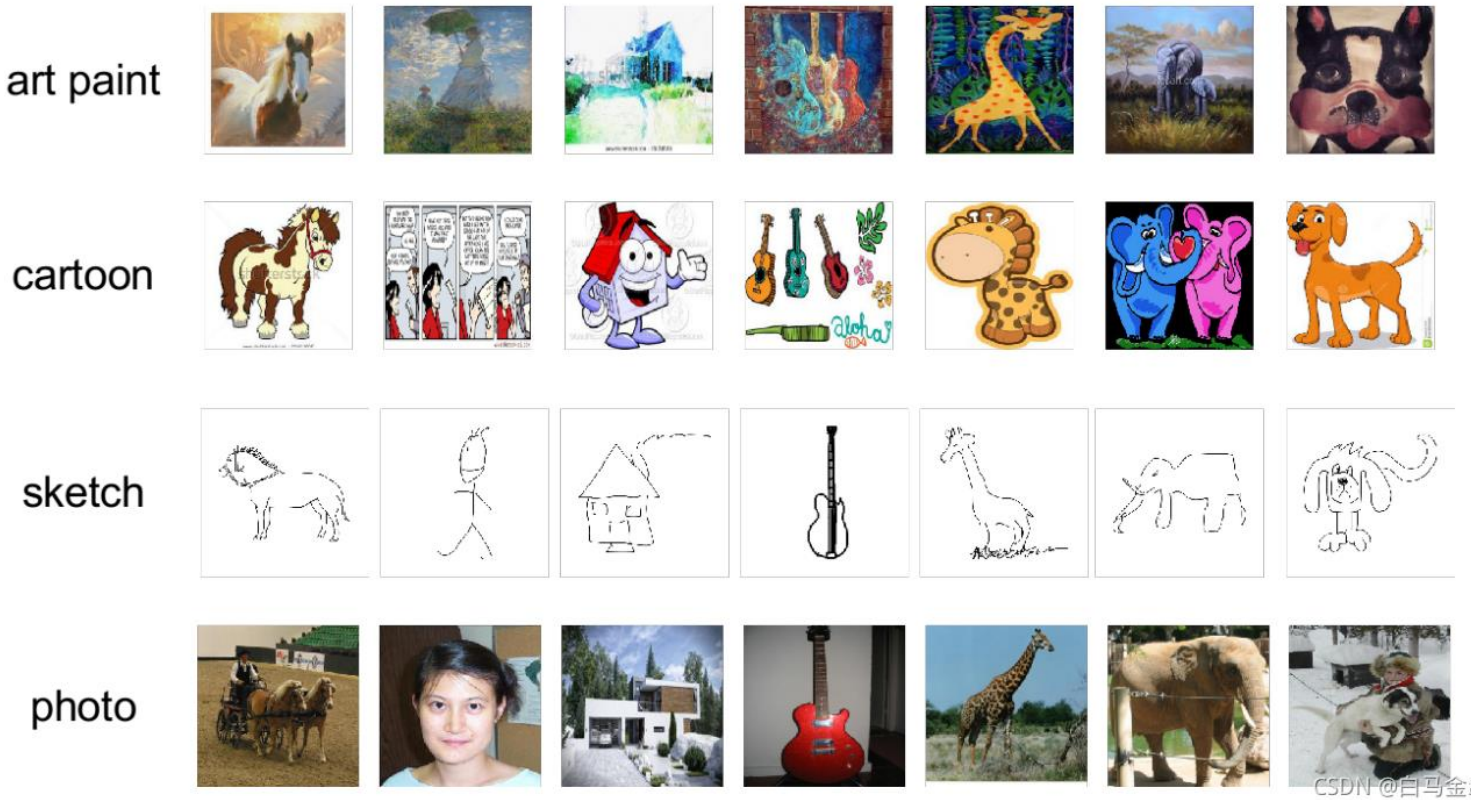
	Best Val Acc
MLP	86.75
GNN	91.76 (↑ 5.01)

The PACS Dataset

PACS is an image dataset for domain generalization.

It consists of four domains, namely
- Photo (1,670 images),
- Art Painting (2,048 images),
- Cartoon (2,344 images) and
- Sketch (3,929 images).

Each domain contains seven categories.



AI VIET NAM
@aivietnam.edu.vn

MLP vs. GNN

Use 1 domain for training

```python
# Clone github repository with data
if not os.path.isdir("./Homework3-PACS"):
    !git clone https://github.com/MachineLearning2020/Homework3-PACS


transf = transforms.Compose(
    [
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225]),
    ]
)

# Define datasets root
DIR_PHOTO = "Homework3-PACS/PACS/photo"
DIR_ART = "Homework3-PACS/PACS/art_painting"
DIR_CARTOON = "Homework3-PACS/PACS/cartoon"
DIR_SKETCH = "Homework3-PACS/PACS/sketch"

# Prepare Pytorch train/test Datasets
photo_dataset = ImageFolder(DIR_PHOTO, transform=transf)
art_dataset = ImageFolder(DIR_ART, transform=transf)
cartoon_dataset = ImageFolder(DIR_CARTOON, transform=transf)
sketch_dataset = ImageFolder(DIR_SKETCH, transform=transf)
```

photo2art

```python
# Split dataset into train and test
train_size = int(0.8 * len(photo_dataset))
test_size = len(photo_dataset) - train_size

train_dataset, test_dataset = torch.utils.data.random_split(
    photo_dataset, [train_size, test_size]
)

# Concatenate all datasets
test_datasets = torch.utils.data.ConcatDataset([test_dataset, art_dataset])

# Create Dataloaders
trainloader = DataLoader(
    train_dataset, batch_size=128, shuffle=True, num_workers=4, drop_last=True
)
testloader = DataLoader(
  test_datasets, batch_size=128, shuffle=False, num_workers=4
)
```

	Best Val Acc
MLP	54.58
GNN	66.46 (↑ 11.8)

MLP vs. GNN

photo2art

	Best Val Acc
MLP	54.58
GNN	66.46 (↑ 11.8)

photo2sketch

	Best Val Acc
MLP	22.12
GNN	29.35 (↑ 7.23)

photo_cartoon2art

	Best Val Acc
MLP	70.15
GNN	85.62 (↑ 15.47)

photo2all

	Best Val Acc
MLP	26.6
GNN	31.44 (↑ 4.48)

AI VIET NAM
@aivietnam.edu.vn

MLP vs. GNN

Use all domains for training and testing

```python
# Clone github repository with data
if not os.path.isdir("./Homework3-PACS"):
    !git clone https://github.com/MachineLearning2020/Homework3-PACS


transf = transforms.Compose(
    [
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225]),
    ]
)

# Define datasets root
DIR_PHOTO = "Homework3-PACS/PACS/photo"
DIR_ART = "Homework3-PACS/PACS/art_painting"
DIR_CARTOON = "Homework3-PACS/PACS/cartoon"
DIR_SKETCH = "Homework3-PACS/PACS/sketch"

# Prepare Pytorch train/test Datasets
photo_dataset = ImageFolder(DIR_PHOTO, transform=transf)
art_dataset = ImageFolder(DIR_ART, transform=transf)
cartoon_dataset = ImageFolder(DIR_CARTOON, transform=transf)
sketch_dataset = ImageFolder(DIR_SKETCH, transform=transf)
```

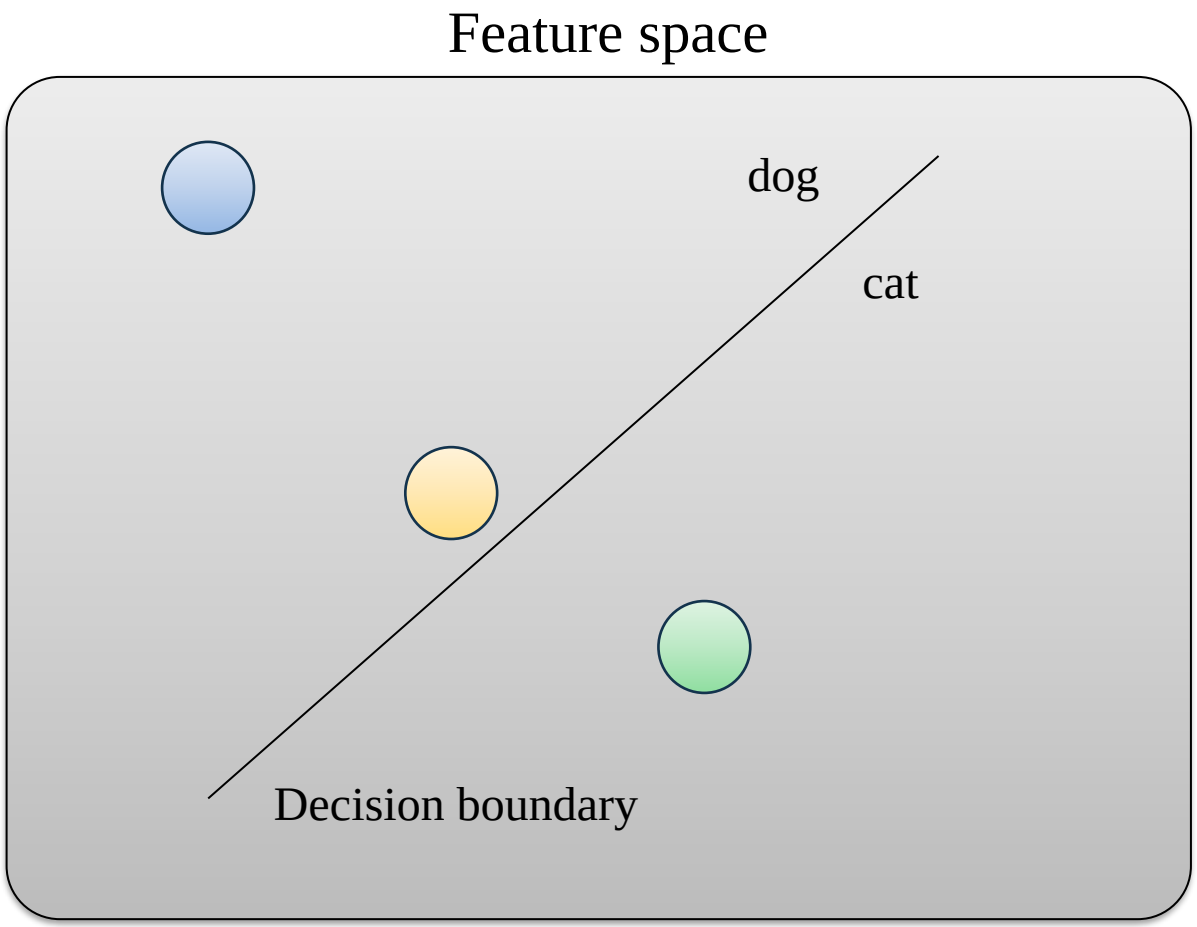
all2all

```python
# concat all datasets
full_dataset = torch.utils.data.ConcatDataset([photo_dataset,
                                               art_dataset,
                                               cartoon_dataset,
                                               sketch_dataset])

# Split dataset into train and test
train_size = int(0.8 * len(full_dataset))
test_size = len(full_dataset) - train_size

train_dataset, test_dataset = random_split(full_dataset, [train_size, test_size])

# Create Dataloaders
trainloader = DataLoader(
    train_dataset, batch_size=128, shuffle=True, num_workers=4, drop_last=True
)
testloader = DataLoader(
  test_datasets, batch_size=128, shuffle=False, num_workers=4
)
```

	Best Val Acc
MLP	90.40
GNN	91.35 (↑ 0.95)

Limitations

- Require clean data
- Require batch size > 1
- Slower training and inference (computational expensive)

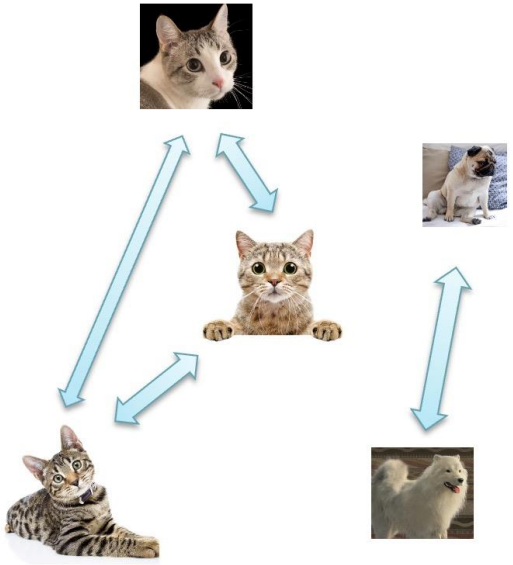


Summary

To utilize GNNs (Graph Neural Networks) for any problem, three sequential questions need to be addressed:

- What is a node?
- What is an edge?
- Can we incorporate an auxiliary loss function?

How does GNN operate?



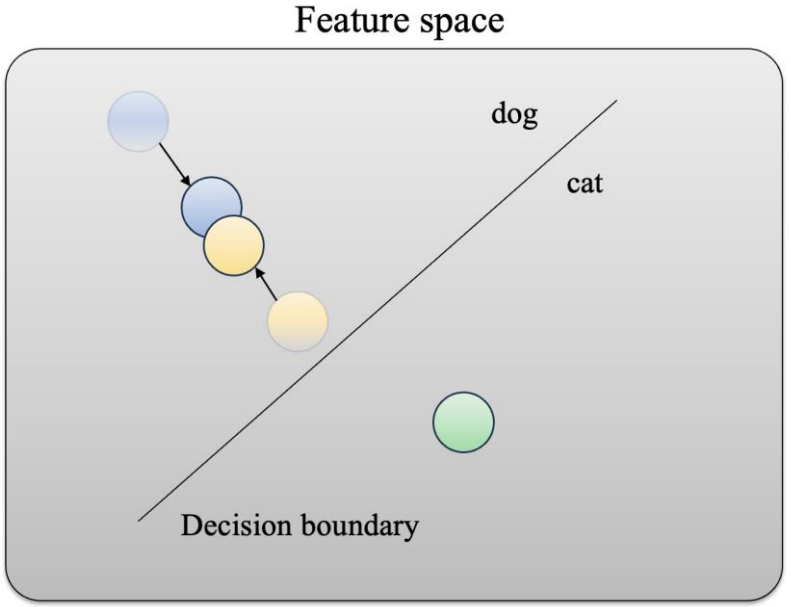


photo2art

	Best Val Acc
MLP	54.58
GNN	66.46 (↑ 11.8)

photo2sketch

	Best Val Acc
MLP	22.12
GNN	29.35 (↑ 7.23)

all2all

	Best Val Acc
MLP	90.40
GNN	91.35 (↑ 0.95)

photo_cartoon2art

	Best Val Acc
MLP	70.15
GNN	85.62 (↑ 15.47)

photo2all

	Best Val Acc
MLP	26.6
GNN	31.44 (↑ 4.48)

Caltech101

	Best Val Acc
MLP	86.75
GNN	91.76 (↑ 5.01)

Thanks!

Any questions?