Edge-Labeling Graph Neural Network for Few-shot Learning (2019 CVPR)

Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D.Yoo

-Korea Advanced Institute of Science and Technology
-MILA, Universite de Montreal
-Kakao Brain

2017-27619

Hyun Jun Choi

Seoul National University

Few shot learning

Amanita muscaria

Amanita muscaria, commonly known as the **fly agaric** or **fly amanita**, is a basidiomycete of the genus *Amanita*. It is also a muscimol mushroom. Native throughout the temperate and boreal regions of the Northern Hemisphere, *Amanita muscaria* has been unintentionally introduced to many countries in the Southern Hemisphere, generally as a symbiont with pine and birch plantations, and is now a true cosmopolitan species. It associates with various deciduous and coniferous trees.

https://en.wikipedia.org/wiki/Amanita_muscaria

- 광대버섯

-독성이 있음!

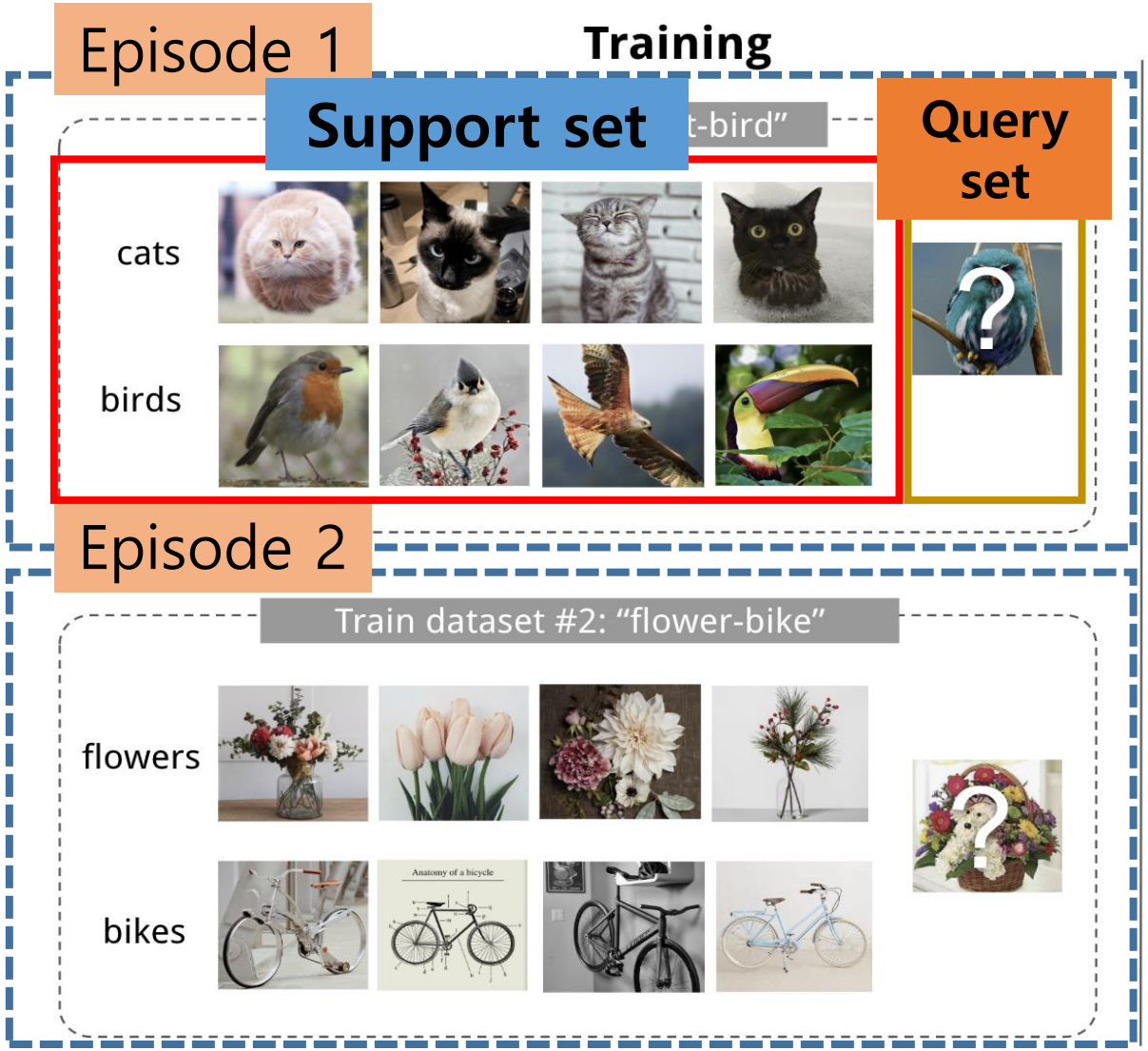
Few shot learning

광대버섯(독성 있음!)

-ML supervised learning 관점에서
-한 개의 data로 그 class에 대한 classifier 학습
-어려운 task

https://en.wikipedia.org/wiki/Amanita_muscaria

Few shot learning (2-way 4 shot learning)



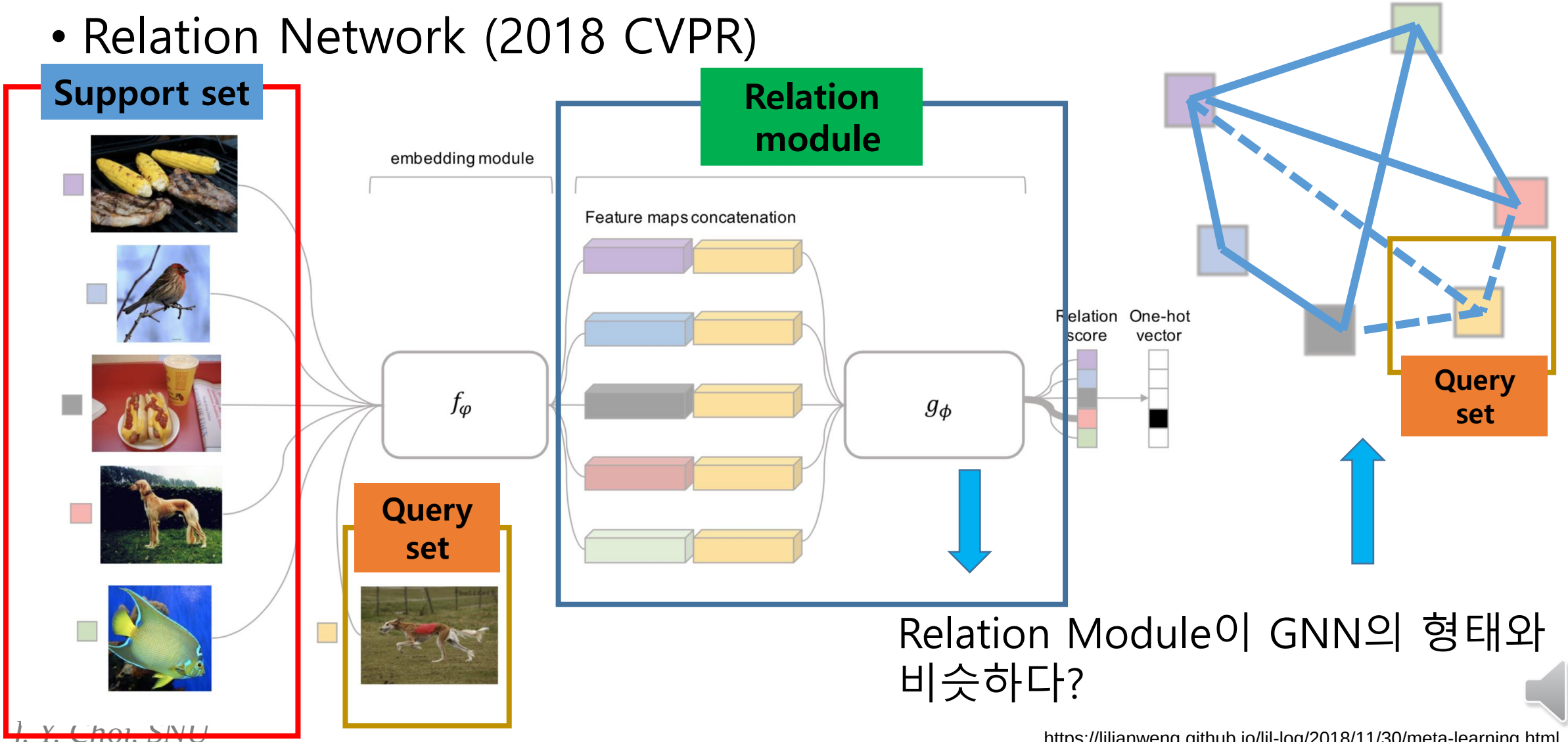
Testing

Train 과정에서 없는 새로운 class

https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html

Metric 기반의 기존 방법 → GNN 기반의 방법

- Relation Network (2018 CVPR)



Relation Module이 GNN의 형태와 비슷하다?

https://lilianweng.github.io/lil-log/2018/11/30/meta-learning.html

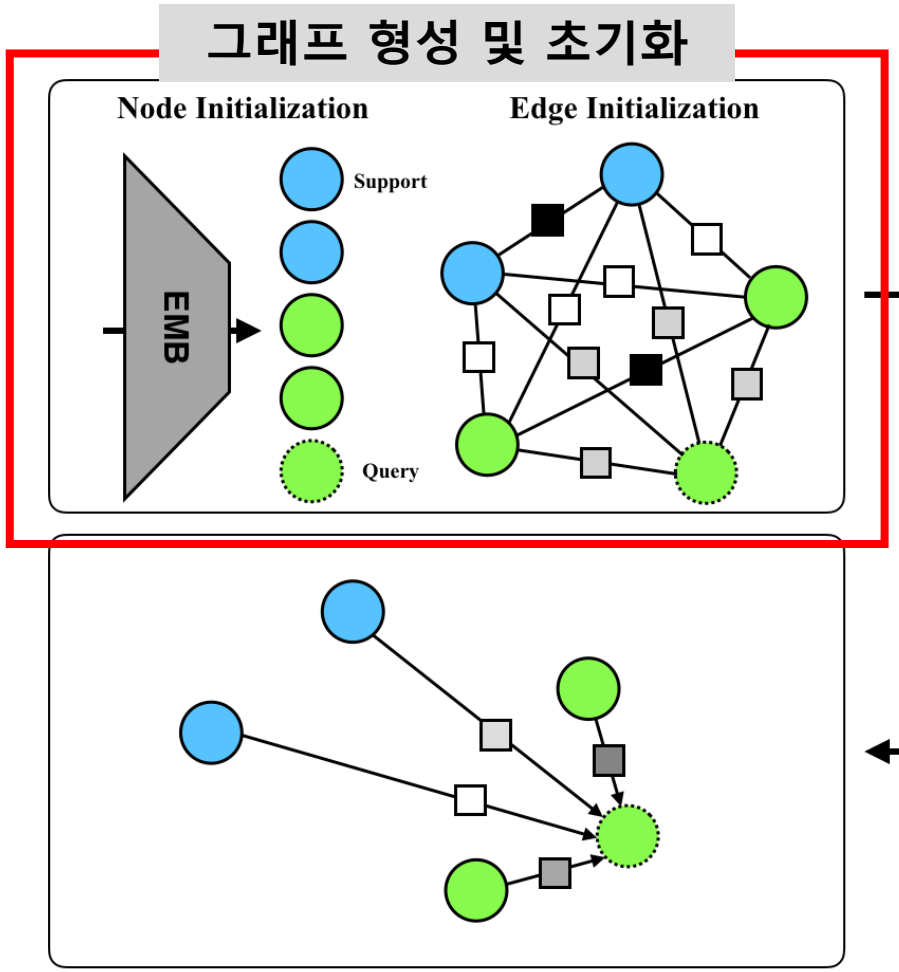
Proposed Method (EGNN)

	범주 정보를 담는 부분	범주별 군집화 여부	업데이트를 하는 부분	훈련 단계에서 범주 수 변경 가능 여부
GNN	노드	X	노드	X
TPN	노드	X	X	O
EGNN	간선	O	노드와 간선	O

- 기존 GNN의 문제점
  - 노드에 대한 업데이트 및 이에 대한 결과로 바로 노드의 class를 얻음
  - 같은 클라스 끼리 근접하게, 다른 클라스끼리 서로 멀어지게 하지 않음
- EGNN에서 제안한 방법
  - Edge를 통해 노드 간의 유사도를 직접적으로(explicitly) 모델링 함
  - Edge의 value를 통해 클라스 끼리 근접하게, 다른 클라스끼리 서로 멀어지게 함

https://www.kakaobrain.com/blog/112

Proposed Method



(c) Query Node Label Prediction

- Node Initialization
  - EMB : Embedding Network 로써, 기존 논문들이 사용하던 단순한 4-block CNN backbone network 사용(image to feature)
- Edge Initialization (같은 class 1, 다르면 0)

$$y_{ij} = \begin{cases} 1, & \text{if } y_i = y_j, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

$$\mathbf{e}_{ij}^0 = \begin{cases} [1||0], & \text{if } y_{ij} = 1 \text{ and } i,j \leq N \times K, \\ [0||1], & \text{if } y_{ij} = 0 \text{ and } i,j \leq N \times K, \\ [0.5||0.5], & \text{otherwise,} \end{cases} \quad (2)$$

  - 2d vector로 intra-cluster similarity, inter-cluster dissimilarity 표현
    ->두 개로 분리한 경우 성능이 제일 좋았음

(b-2) EGNN: Edge Feature Update

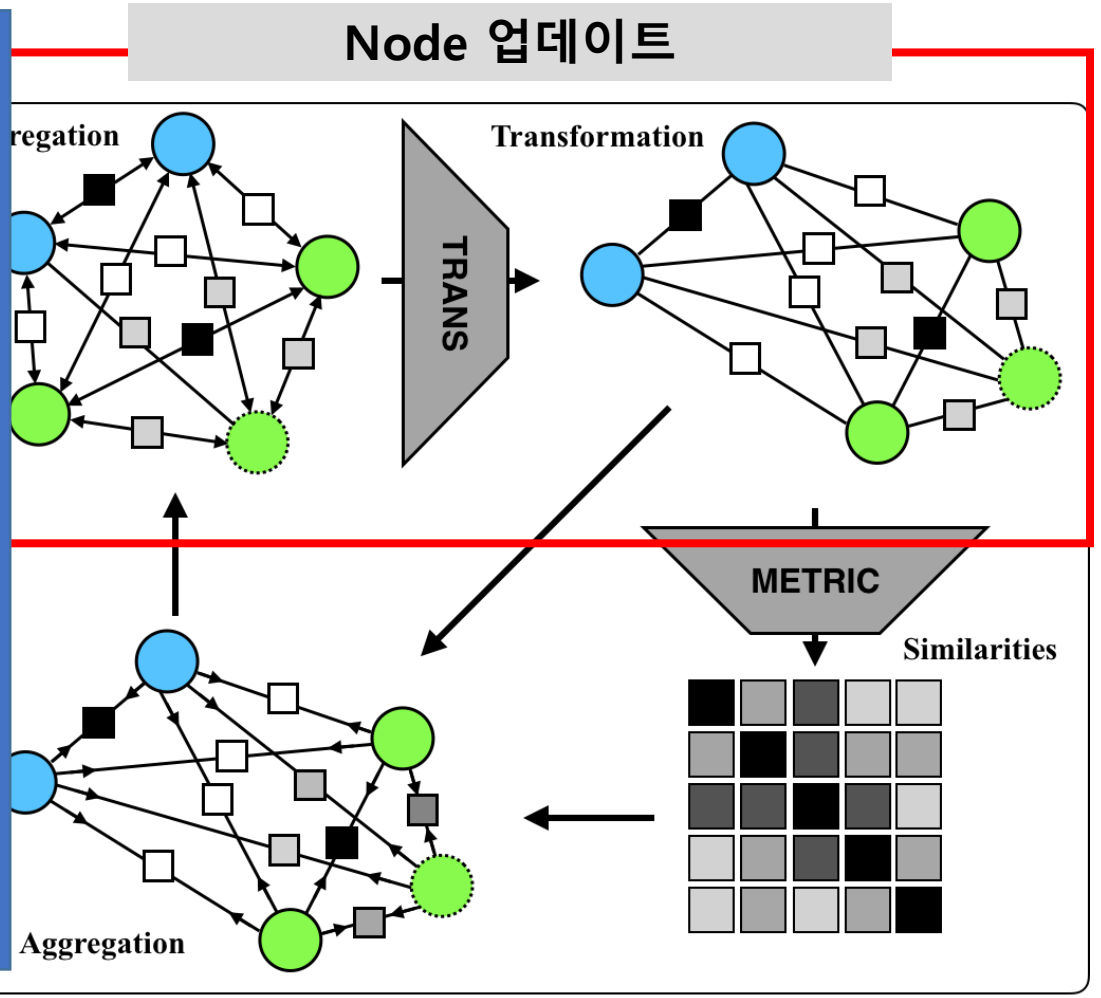
https://www.kakaobrain.com/blog/112

Proposed Method

- Node feature update
  - TRANS: 이전의 edge와 node feature로부터 node feature update(1-hop) 함
  - Attention mechanism과 유사한 원리 이용
  - 대신, Edge feature가 곧 Attention 에 대한 degree of contribution

$$\mathbf{v}_i^{\ell} = f_v^{\ell}([\sum_j \tilde{e}_{ij1}^{\ell-1}\mathbf{v}_j^{\ell-1} || \sum_j \tilde{e}_{ij2}^{\ell-1}\mathbf{v}_j^{\ell-1}]; \theta_v^{\ell}),$$

$$\tilde{e}_{ijd} = \frac{e_{ijd}}{\sum_k e_{ikd}}$$



(c) Query Node Label Prediction

(b-2) EGNN: Edge Feature Update

https://www.kakaobrain.com/blog/112

Proposed Method

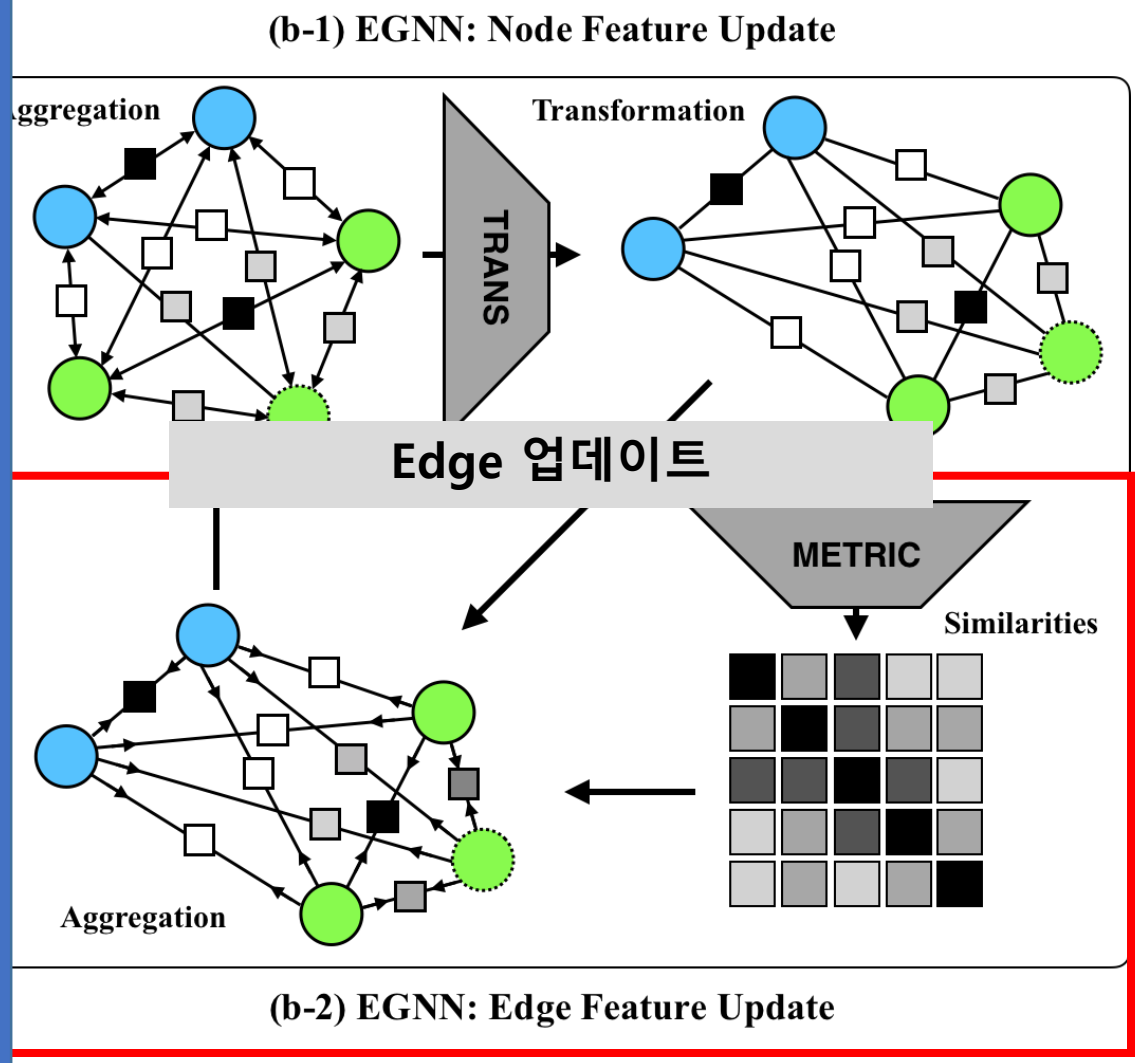
- Edge feature update
  - 업데이트 된 node feature 기반으로 노드 간의 연결성(metric 네트워크)을 구함
  - node feature간의 metric(연결성) +이전의 edge feature 둘을 결합하여 새로이 edge feature update 함

$$\bar{e}_{ij1}^{\ell} = \frac{f_e^{\ell}(\mathbf{v}_i^{\ell}, \mathbf{v}_j^{\ell}; \theta_e^{\ell}) e_{ij1}^{\ell-1}}{\sum_k f_e^{\ell}(\mathbf{v}_i^{\ell}, \mathbf{v}_k^{\ell}; \theta_e^{\ell}) e_{ik1}^{\ell-1} / (\sum_k e_{ik1}^{\ell-1})},$$

$$\bar{e}_{ij2}^{\ell} = \frac{(1 - f_e^{\ell}(\mathbf{v}_i^{\ell}, \mathbf{v}_j^{\ell}; \theta_e^{\ell})) e_{ij2}^{\ell-1}}{\sum_k (1 - f_e^{\ell}(\mathbf{v}_i^{\ell}, \mathbf{v}_k^{\ell}; \theta_e^{\ell})) e_{ik2}^{\ell-1} / (\sum_k e_{ik2}^{\ell-1})},$$

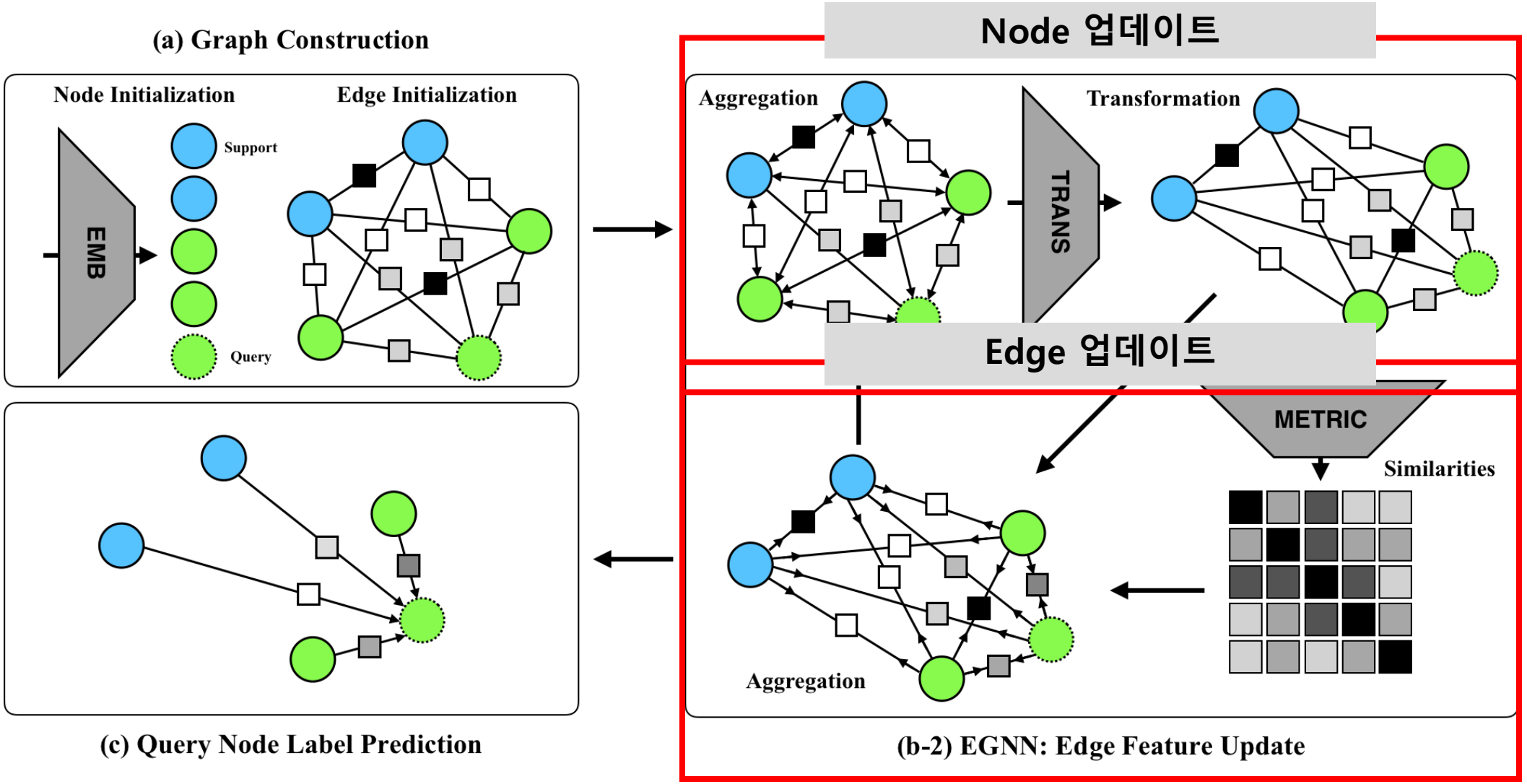
$$\mathbf{e}_{ij}^{\ell} = \bar{\mathbf{e}}_{ij}^{\ell} / \|\bar{\mathbf{e}}_{ij}^{\ell}\|_1,$$

f_e^{ℓ} is the metric network

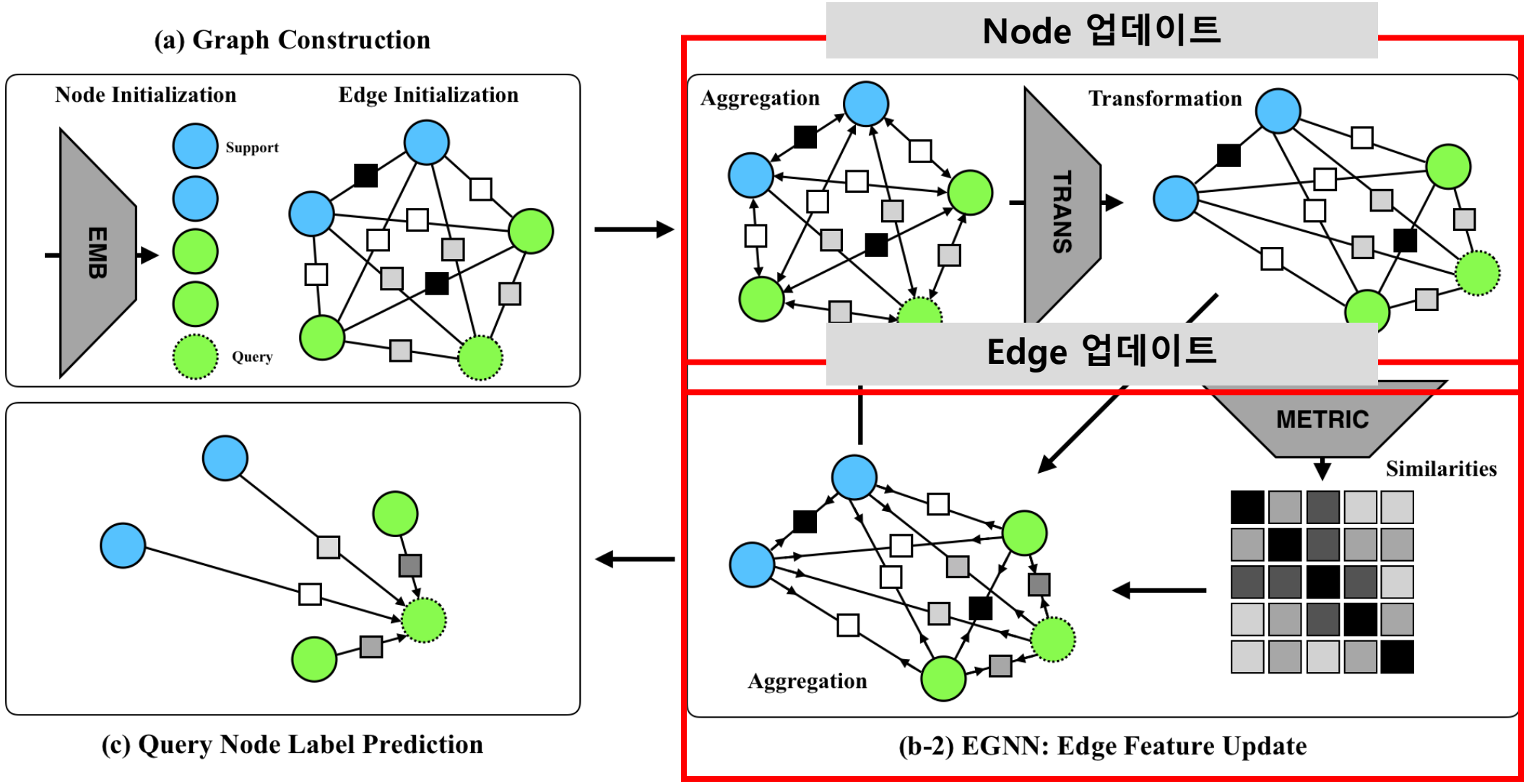


https://www.kakaobrain.com/blog/112

Proposed Method

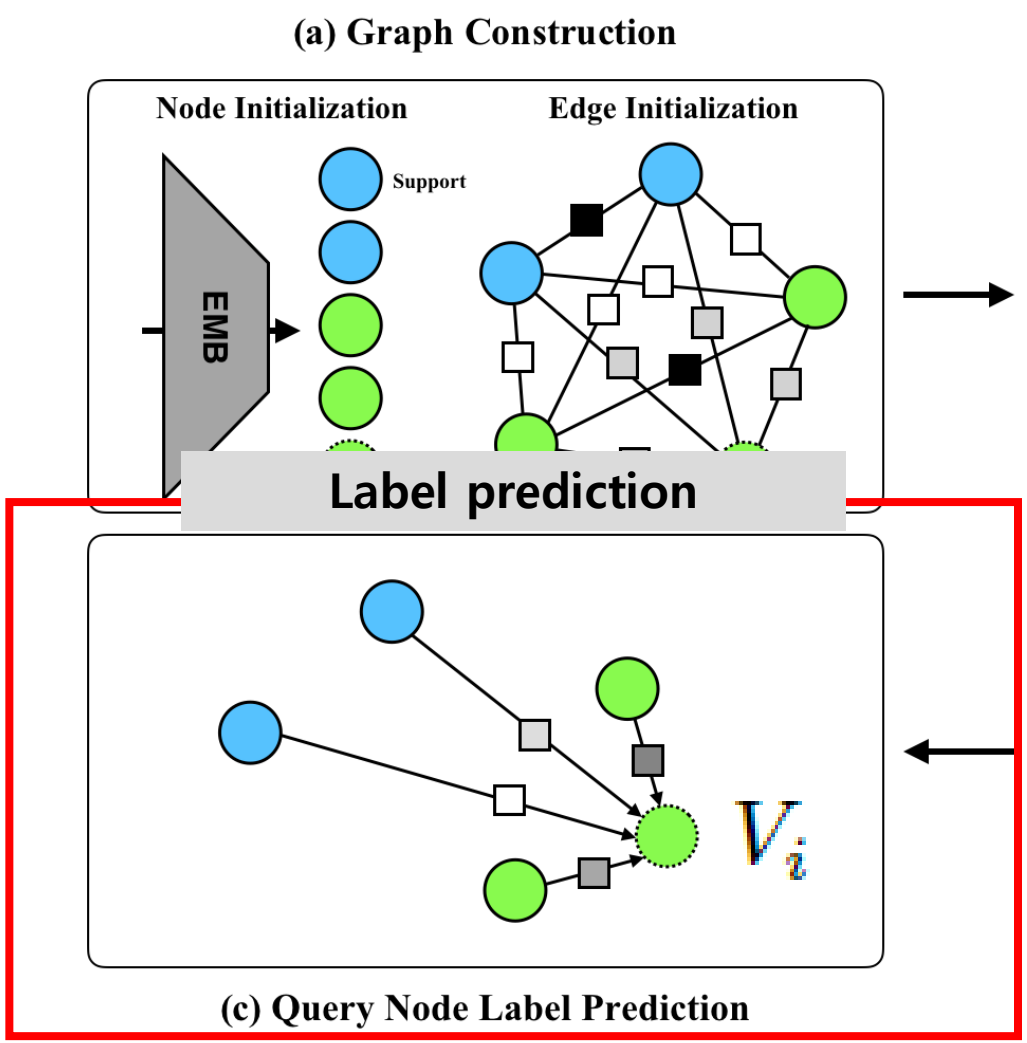


Proposed Method



https://www.kakaobrain.com/blog/112

Proposed Method



- Label prediction
  - 인접한 노드 중 label이 있는 것으로 부터 voting을 통해 결정
  - Voting에 대한 가중치는 연결된 edge, 즉 similarity가 결정
  - 최종적으로 class 별 similarity에 대해 softmax를 취해 확률로 표현

V_i 가 C_k class 일 확률

$$P(y_i = \mathcal{C}_k | \mathcal{T}) = p_i^{(k)}$$

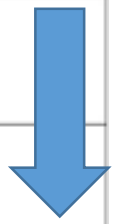
$$p_i^{(k)} = \text{softmax}\left(\sum_{\{j:j \neq i \wedge (\mathbf{x}_j, y_j) \in \mathcal{S}\}} \hat{y}_{ij} \delta(y_j = \mathcal{C}_k) \right)$$

(b-2) EGNN: Edge Feature Update

https://www.kakaobrain.com/blog/112

데이터 셋

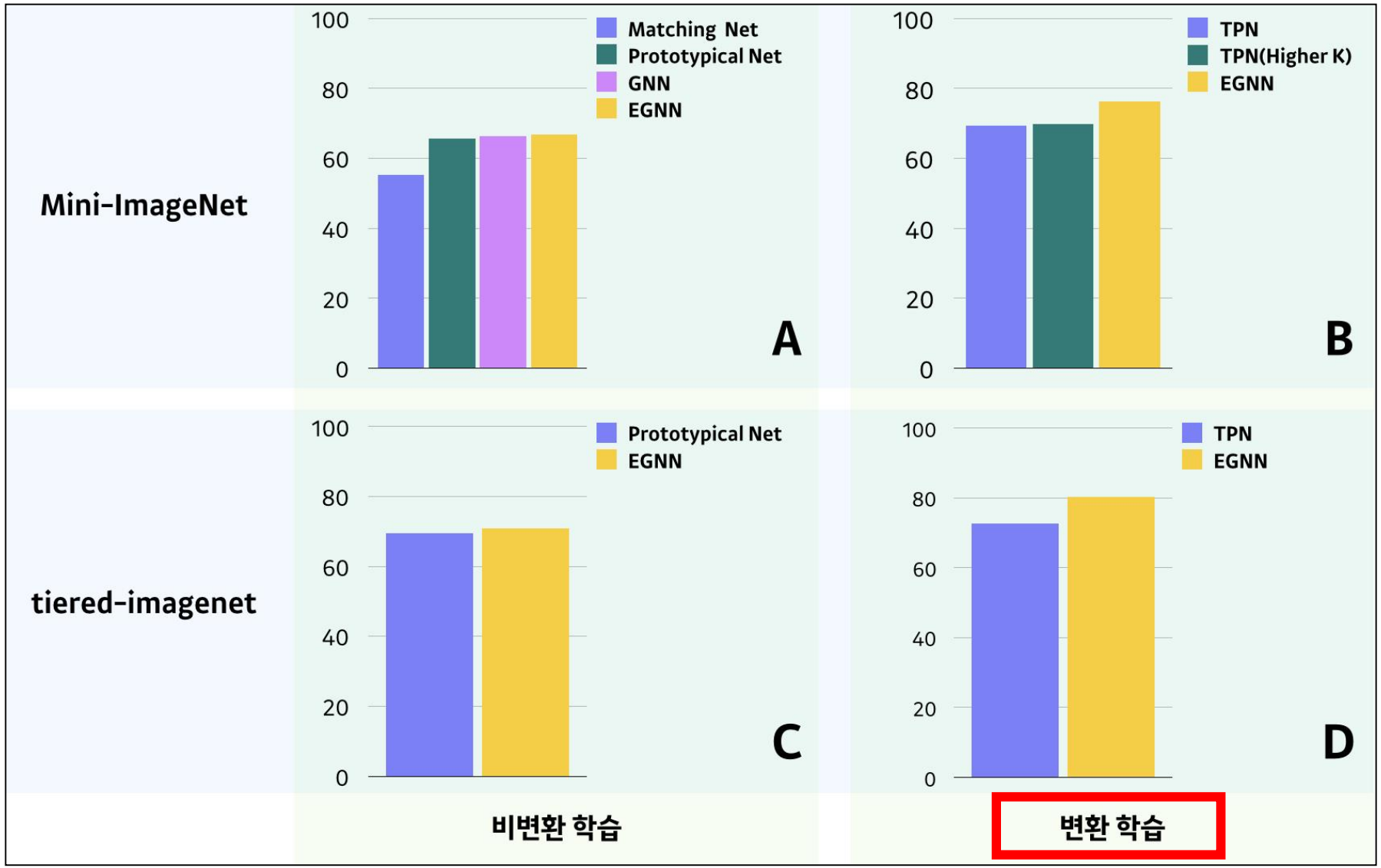
데이터셋 이름	데이터 속성	범주 수	범주 별 이미지 수	총 데이터	훈련/검증/테스트 범주수
mini-imagenet	84×84 RGB 이미지	100	600	60,000	60/16/20
tiered-imagenet		608	약 1,300	779,165	351/97/160



데이터 수 및 난이도 증가

- 무작위로 샘플링하여 600개의 N-way K-shot 테스트 태스크 구성
- 모든 쿼리 데이터에 대한 범주 예측 정확도 (범주를 맞춘 쿼리 수/전체 쿼리 수)로 평가

성능 평가 (supervised)



Supervised learning에 있어서

기존의 GNN들에 비해 성능 향상

https://www.kakaobrain.com/blog/112

성능 평가 (semi-supervised)

- semi-supervised

모델	학습 방식		서포트 데이터에서 라벨링 데이터 비율			
			20%	40%	60%	100%
GNN	감독학습	비변환학습	50.33	56.91	-	66.41
	준감독학습		52.45	58.76	-	
EGNN	감독학습		52.86		-	66.85
	준감독학습		61.88	62.52	63.53	
	감독학습	변환학습	59.18	-	-	75.37
	준감독학습		63.62	64.32	66.37	

Ablation study

- 훈련과 테스트 과정에서 범주 수가 바뀌어도 유연하게 대처

모델	훈련 시 범주 수	테스트 시 범주 수	정확도
GNN	5	5	66.41
		10	N/A
	10	10	51.75
		5	N/A
EGNN	5	5	75.37
		10	56.35
	10	10	57.61
		5	76.27

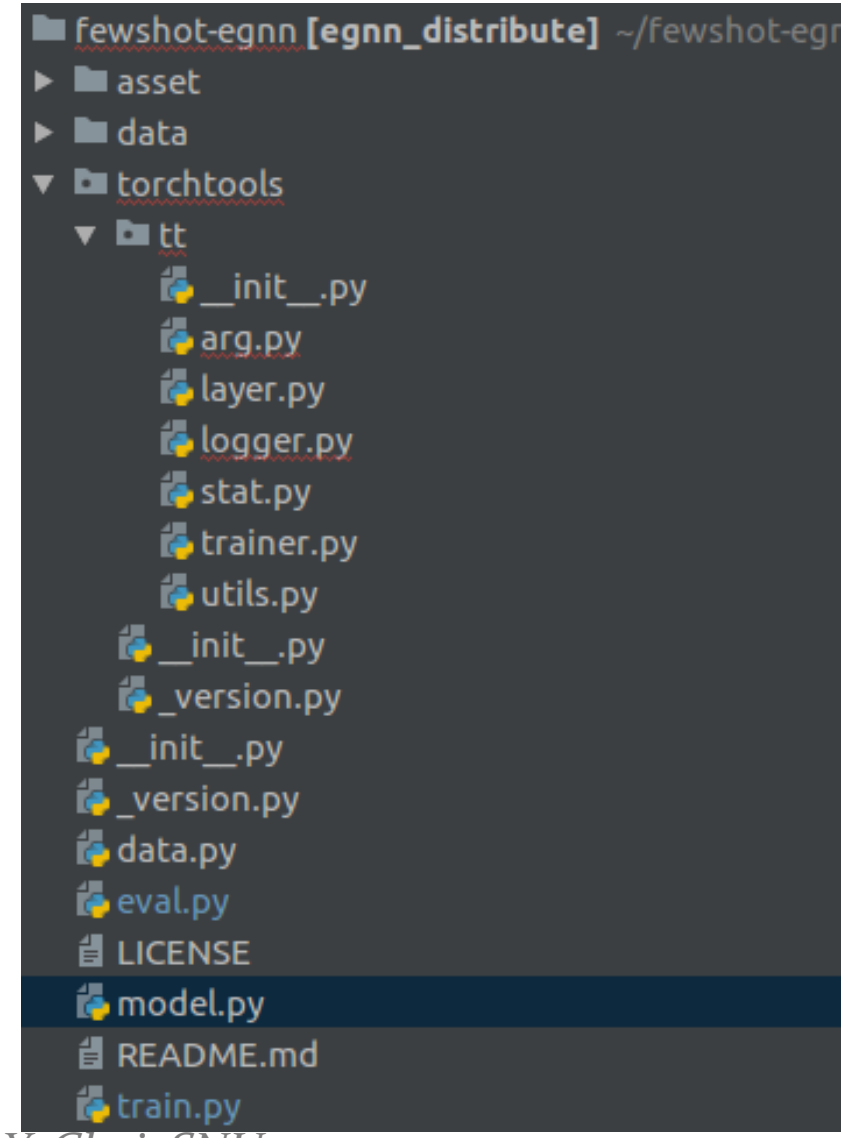
결론 및 논의

- Few shot learning 분야에서 GNN 적용 시 좋은 성능 얻음
- GNN에 있어서 단순 node 뿐 아니라 edge에 대한 modeling 필요
- Edge 에 대한 modeling 시 성능 및 유연성 증가
- 향후 GNN의 application분야에서 node 뿐 아니라 edge에 대한 modeling이 중요할 것으로 보임

실험

- 공개된 코드를 바탕으로 구체적인 이해 및 실험 수행
- 공개된 코드를 통해 학습하여 논문과 거의 비슷한 성능을 도출
- 공개된 코드에 대해 더 자세하게 설명 주석 추가 및 업데이트
- 업데이트 된 코드 링크 : https://github.com/hyunjunChhoi/fewshot-egnn

코드의 구성

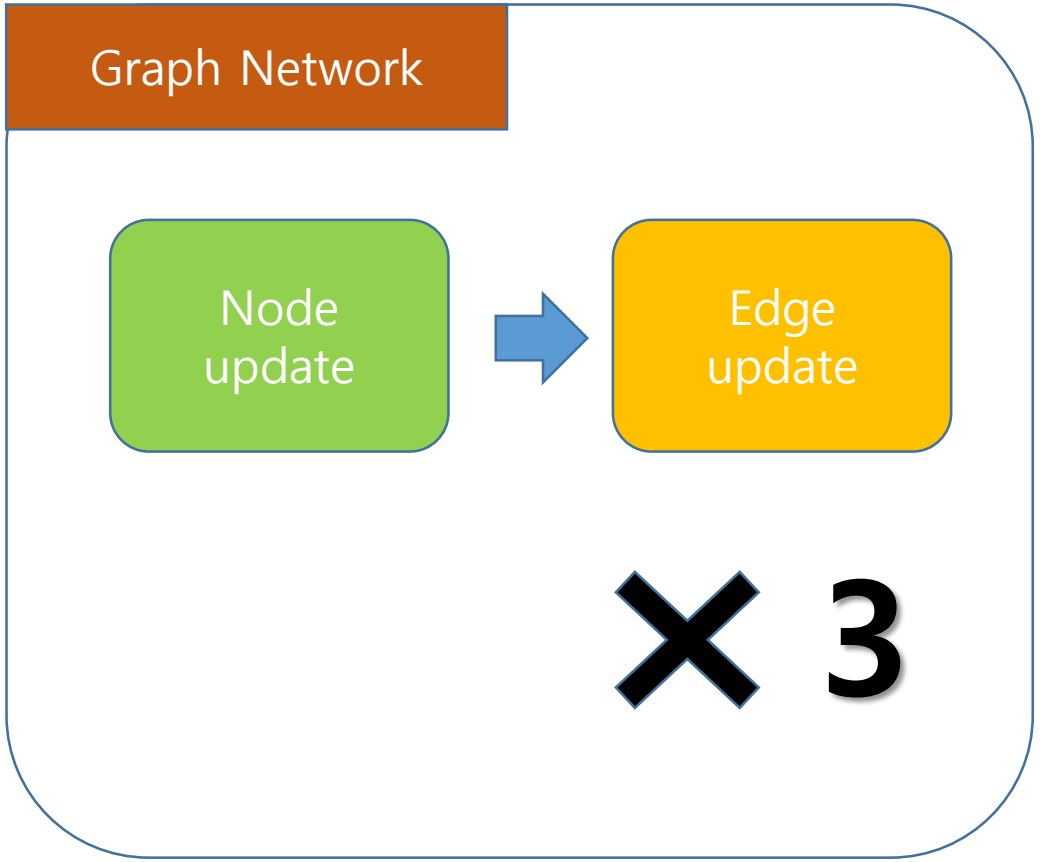


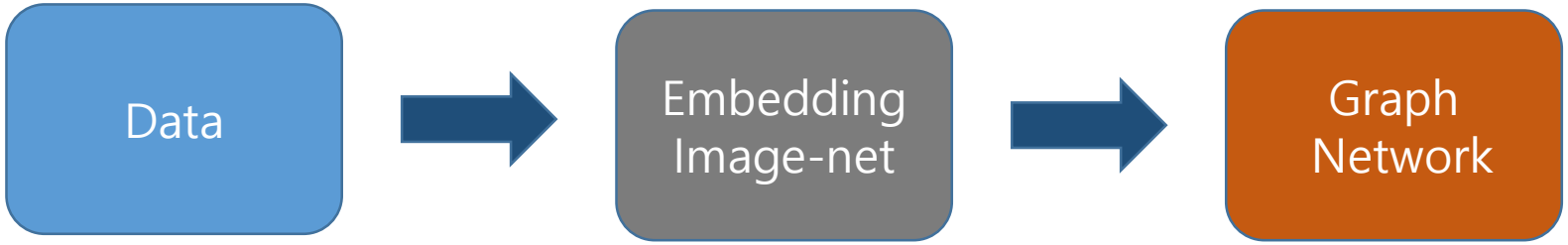
- Asset : 훈련 및 체크 포인트 저장 위치
- Data : mini-imagenet, tired-imagenet 데이터 저장 장소
- Torchtools
  - 잡다한 유틸리티 저장소
  - Logger 는 기록저장
  - 기타 실행 시간 측정 또는 체크 포인트 로딩 등등의 기능

- Data.py : 실질적인 data-loader를 구성하는 부분
- Eval.py : test 시에 쓰이는 데, 실질적으로 train.py 에서 구성된 test mode를 실행
- Train.py : train 및 test시에 모두 사용됨 대부분의 내용을 담고 있는 코드
- Model.py : EGNN 의 모델을 전부 담고 있는 코드

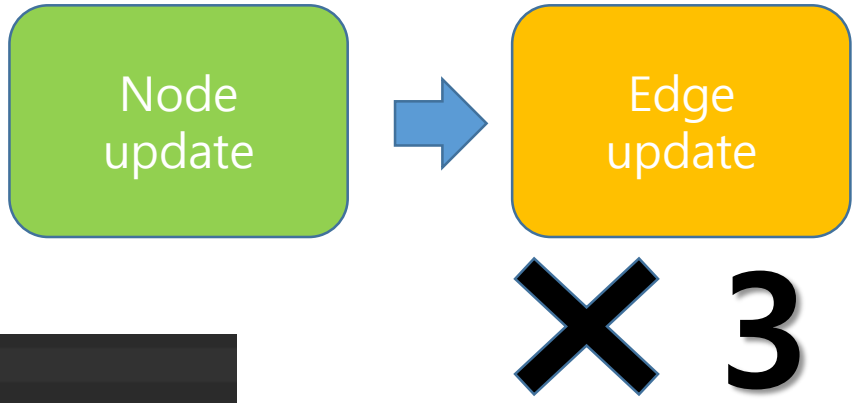
EGNN 모델

- Model.py
  - Graph Network
  - Node update Network
  - Edge update Network
  - Embedding image-net (backbone network)





EGNN 모델



- Graph Network

```
# for each layer
for l in range(self.num_layers):
    # set edge to node
    edge2node_net = NodeUpdateNetwork(in_features=self.in_features if l == 0 else self.node_features,
                                      num_features=self.node_features,
                                      dropout=self.dropout if l < self.num_layers-1 else 0.0)

    # set node to edge
    node2edge_net = EdgeUpdateNetwork(in_features=self.node_features,
                                      num_features=self.edge_features,
                                      separate_dissimilarity=False,
                                      dropout=self.dropout if l < self.num_layers-1 else 0.0)

    self.add_module('edge2node_net{}'.format(l), edge2node_net)
    self.add_module('node2edge_net{}'.format(l), node2edge_net)
```

  - Node network 와 Edge network 를 num_layers 수 만큼 반복 추가하여 구성함
  - 논문 및 코드에서는 num_layers 를 3으로 함

EGNN 모델

- Embedding image-net (backbone network)
  - 논문에서 언급하였듯이, 이전 논문 GNN과 같은 Backbone network로 구성
  - Leaky Relu를 쓴 점이 특이
  - Dropout과 Batch-norm 모두 사용

```
self.conv_1 = nn.Sequential(nn.Conv2d(in_channels=3,
                                      out_channels=self.hidden,
                                      kernel_size=3,
                                      padding=1,
                                      bias=False),
                            nn.BatchNorm2d(num_features=self.hidden),
                            nn.MaxPool2d(kernel_size=2),
                            nn.LeakyReLU(negative_slope=0.2, inplace=True))
self.conv_2 = nn.Sequential(nn.Conv2d(in_channels=self.hidden,
                                      out_channels=int(self.hidden*1.5),
                                      kernel_size=3,
                                      bias=False),
                            nn.BatchNorm2d(num_features=int(self.hidden*1.5)),
                            nn.MaxPool2d(kernel_size=2),
                            nn.LeakyReLU(negative_slope=0.2, inplace=True))
self.conv_3 = nn.Sequential(nn.Conv2d(in_channels=int(self.hidden*1.5),
                                      out_channels=self.hidden*2,
                                      kernel_size=3,
                                      padding=1,
                                      bias=False),
                            nn.BatchNorm2d(num_features=self.hidden * 2),
                            nn.MaxPool2d(kernel_size=2),
                            nn.LeakyReLU(negative_slope=0.2, inplace=True),
                            nn.Dropout2d(0.4))
self.conv_4 = nn.Sequential(nn.Conv2d(in_channels=self.hidden*2,
                                      out_channels=self.hidden*4,
                                      kernel_size=3,
                                      padding=1,
                                      bias=False),
                            nn.BatchNorm2d(num_features=self.hidden * 4),
                            nn.MaxPool2d(kernel_size=2),
                            nn.LeakyReLU(negative_slope=0.2, inplace=True),
                            nn.Dropout2d(0.5))
self.layer_last = nn.Sequential(nn.Linear(in_features=self.last_hidden * 4,
                                          out_features=self.emb_size, bias=True),
                                nn.BatchNorm1d(self.emb_size))
```

EGNN 모델

$$\tilde{e}_{ijd} = \frac{e_{ijd}}{\sum_k e_{ikd}} \qquad \mathbf{v}_i^{\ell} = f_v^{\ell}([\sum_j \tilde{e}_{ij1}^{\ell-1}\mathbf{v}_j^{\ell-1} \| \sum_j \tilde{e}_{ij2}^{\ell-1}\mathbf{v}_j^{\ell-1}]; \theta_v^{\ell}),$$

- Node network

```
def forward(self, node_feat, edge_fe  ):
    # get size
    num_tasks = node_feat.size(0)
    num_data = node_feat.size(1)

    # get eye matrix (batch_size x 2  x node_size x node_size)
    diag_mask = 1.0 - torch.eye(num_  ta).unsqueeze(0).unsqueeze(0).repeat(num_tasks  , 1, 1).to(tt.arg.device)

    # set diagonal as zero and norma ize
    edge_feat = F.normalize(edge_feat * diag_mask, p=1, dim=-1)

    # compute attention and aggregate
    aggr_feat = torch.bmm(torch.cat(torch.split(edge_feat, 1, 1), 2).squeeze(1), no  eat)

    node_feat = torch.cat([node_feat, torch.cat(aggr_feat.split(num_data, 1), -1)], -1).transpose(1, 2)

    # non-linear transform
    node_feat = self.network(node_feat.unsqueeze(-1)).transpose(1, 2).squeeze(-1)
    return node_feat
```

EGNN 모델

- Edge network

$$
\begin{aligned}
\bar{e}^{\ell}_{ij1} &= \frac{f^{\ell}_e(\mathbf{v}^{\ell}_i, \mathbf{v}^{\ell}_j; \theta^{\ell}_e) e^{\ell-1}_{ij1}}{\sum_k f^{\ell}_e(\mathbf{v}^{\ell}_i, \mathbf{v}^{\ell}_k; \theta^{\ell}_e) e^{\ell-1}_{ik1} / (\sum_k e^{\ell-1}_{ik1})}, \\
\bar{e}^{\ell}_{ij2} &= \frac{(1 - f^{\ell}_e(\mathbf{v}^{\ell}_i, \mathbf{v}^{\ell}_j; \theta^{\ell}_e)) e^{\ell-1}_{ij2}}{\sum_k (1 - f^{\ell}_e(\mathbf{v}^{\ell}_i, \mathbf{v}^{\ell}_k; \theta^{\ell}_e)) e^{\ell-1}_{ik2} / (\sum_k e^{\ell-1}_{ik2})}, \\
\mathbf{e}^{\ell}_{ij} &= \bar{\mathbf{e}}^{\ell}_{ij} / \|\bar{\mathbf{e}}^{\ell}_{ij}\|_1,
\end{aligned}
$$

f^{ℓ}_e is the metric network

```
def forward(self, node_feat, edge_fea  :
    # compute abs(x_i, x_j)
    x_i = node_feat.unsqueeze(2)
    x_j = torch.transpose(x_i, 1, 2)
    x_ij = torch.abs(x_i - x_j)
    x_ij = torch.transpose(x_ij, 1, 3

    # compute similarity/dissimilarity (batch_size x feat_size x num_samples x num_samples)
    sim_val = F.sigmoid(self.sim_network(x_ij))

    if self.separate_dissimilarity:
        dsim_val = F.sigmoid(self.dsim_network(x_ij))
    else:
        dsim_val = 1.0 - sim_val
```

Feature 간의 차이의 절대값 x_ij를 input으로 하는 similarity network라는 Metric network를 거치고 sigmoid를 취함 (dsim 은 dissimilarity network)

EGNN 모델

- Edge network

f_e^ℓ is the metric network

$$\bar{e}_{ij1}^\ell = \frac{f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_j^\ell; \theta_e^\ell) e_{ij1}^{\ell-1}}{\sum_k f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_k^\ell; \theta_e^\ell) e_{ik1}^{\ell-1} / (\sum_k e_{ik1}^{\ell-1})},$$

$$\bar{e}_{ij2}^\ell = \frac{(1 - f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_j^\ell; \theta_e^\ell)) e_{ij2}^{\ell-1}}{\sum_k (1 - f_e^\ell(\mathbf{v}_i^\ell, \mathbf{v}_k^\ell; \theta_e^\ell)) e_{ik2}^{\ell-1} / (\sum_k e_{ik2}^{\ell-1})},$$

$$\mathbf{e}_{ij}^\ell = \bar{\mathbf{e}}_{ij}^\ell / \|\bar{\mathbf{e}}_{ij}^\ell\|_1,$$

```
diag_mask = 1.0 - torch.eye(node_feat.size(1)).unsqueeze(0).unsquee  (0).repeat(node_  at.size(0), 2, 1, 1).to(tt.a
edge_feat = edge_feat * diag_mask
merge_sum = torch.sum(edge_feat, -1, True)
# set diagonal as zero and normalize
edge_feat = F.normalize(torch.cat([sim_val, dsim_val], 1) * edge_fe  , p=1, dim=-1) * merge_sum
force_edge_feat = torch.cat((torch.eye(node_feat.size(1)).unsqueeze  ), torch.zeros(node_feat.size(1), node_feat.si
edge_feat = edge_feat + force_edge_feat
edge_feat = edge_feat + 1e-6
edge_feat = edge_feat / torch.sum(edge_feat, dim=1).unsqueeze(1).repeat(1, 2, 1, 1)

return edge_feat
```

EGNN 훈련 과정

- train.py
  - Set edge mask (to distinguish support and query edges)
  - For semi-supervised setting, unlabeled data setting
  - Load task data list
  - Edge initialization
  - Encode data
  - Predict edge logit
  - Compute loss
  - Update model
  - Evaluation

EGNN 훈련 과정

- train.py
  - Set edge mask (to distinguish support and query edges)

```
def train(self):
    val_acc = self.val_acc

    # set edge mask (to distinguish support and query edges)
    num_supports = tt.arg.num_ways_train * tt.arg.num_shots_train
    num_queries = tt.arg.num_ways_train * 1
    num_samples = num_supports + num_queries
    support_edge_mask = torch.zeros(tt.arg.meta_batch_size, num_samples, num_samples).to(tt.arg.device)
    support_edge_mask[:, :num_supports, :num_supports] = 1
    query_edge_mask = 1 - support_edge_mask
```

EGNN 훈련 과정

- train.py
  - For semi-supervised setting, unlabeled data setting

```
# for semi-supervised setting, ignore unlabeled support sets for evaluation
for c in range(tt.arg.num_ways_train):
    evaluation_mask[:,
    ((c + 1) * tt.arg.num_shots_train - tt.arg.num_unlabeled):(c + 1) * tt.arg.num_shots_train,
    :num_supports] = 0
    evaluation_mask[:, :num_supports,
    ((c + 1) * tt.arg.num_shots_train - tt.arg.num_unlabeled):(c + 1) * tt.arg.num_shots_train] = 0
```

EGNN 훈련 과정

- train.py
  - Load task data list

```
for iter in range(self.global_step + 1, tt.arg.train_iteration + 1):
    # init grad
    self.optimizer.zero_grad()

    # set current step
    self.global_step = iter

    # load task data list
    [support_data,
     support_label,
     query_data,
     query_label] = self.data_loader['train'].get_task_batch(num_tasks=tt.arg.meta_batch_size,
                                                             num_ways=tt.arg.num_ways_train,
                                                             num_shots=tt.arg.num_shots_train,
                                                             seed=iter + tt.arg.seed)


    # set as single data
    full_data = torch.cat([support_data, query_data], 1)
    full_label = torch.cat([support_label, query_label], 1)
    full_edge = self.label2edge(full_label)
```

EGNN 훈련 과정

- train.py
  - Edge initialization
  - 같은 클라스 사이 edge는 [1,0] 모르는 것은 [0.5,0.5]로 초기화

```
# set init edge
init_edge = full_edge.clone()  # batch_size x 2 x num_samples x num_samples
init_edge[:, :, num_supports:, :] = 0.5
init_edge[:, :, :, num_supports:] = 0.5
for i in range(num_queries):
    init_edge[:, 0, num_supports + i, num_supports + i] = 1.0
    init_edge[:, 1, num_supports + i, num_supports + i] = 0.0

# for semi-supervised setting,
for c in range(tt.arg.num_ways_train):
    init_edge[:, :, ((c+1) * tt.arg.num_shots_train - tt.arg.num_unlabeled):(c+1) * tt.arg.num_shots_train, :num
    init_edge[:, :, :num_supports, ((c+1) * tt.arg.num_shots_train - tt.arg.num_unlabeled):(c+1) * tt.arg.num_sh
```

EGNN 훈련 과정

- train.py
  - Edge initialization

```
# set init edge
init_edge = full_edge.clone()  # batch_size x 2 x num_samples x num_samples
init_edge[:, :, num_supports:, :] = 0.5
init_edge[:, :, :, num_supports:] = 0.5
 for i in range(num_queries):
    init_edge[:, 0, num_supports + i, num_supports + i] = 1.0
    init_edge[:, 1, num_supports + i, num_supports + i] = 0.0

 # for semi-supervised setting,
 for c in range(tt.arg.num_ways_train):
    init_edge[:, :, ((c+1) * tt.arg.num_shots_train - tt.arg.num_unlabeled):(c+1) * tt.arg.num_shots_train, :num
    init_edge[:, :, :num_supports, ((c+1) * tt.arg.num_shots_train - tt.arg.num_unlabeled):(c+1) * tt.arg.num_sh
```

EGNN 훈련 과정

- train.py
  - Encode data

```
# set as train mode
self.enc_module.train()
self.gnn_module.train()

# (1) encode data
full_data = [self.enc_module(data.squeeze(1)) for data in full_data.chunk(full_data.size(1), dim=1)]
full_data = torch.stack(full_data, dim=1) # batch_size x num_samples x featdim
```

EGNN 훈련 과정

- train.py
  - Predict edge logit

```
# (2) predict edge logit (consider only the last layer logit, num_tasks x 2 x num_samples x num_samples)
if tt.arg.train_transductive:
    full_logit_layers = self.gnn_module(node_feat=full_data, edge_feat=init_edge)
else:
    evaluation_mask[:, num_supports:, num_supports:] = 0 # ignore query-query edges, since it is non-transductive setting
    # input_node_feat: (batch_size x num_queries) x (num_support + 1) x featdim
    # input_edge_feat: (batch_size x num_queries) x 2 x (num_support + 1) x (num_support + 1)
    support_data = full_data[:, :num_supports] # batch_size x num_support x featdim
    query_data = full_data[:, num_supports:] # batch_size x num_query x featdim
    support_data_tiled = support_data.unsqueeze(1).repeat(1, num_queries, 1, 1) # batch_size x num_queries x num_support x featdim
    support_data_tiled = support_data_tiled.view(tt.arg.meta_batch_size * num_queries, num_supports, -1) # (batch_size x num_queries) x num_support
    query_data_reshaped = query_data.contiguous().view(tt.arg.meta_batch_size * num_queries, -1).unsqueeze(1) # (batch_size x num_queries) x 1 x fe
    input_node_feat = torch.cat([support_data_tiled, query_data_reshaped], 1) # (batch_size x num_queries) x (num_support + 1) x featdim

    input_edge_feat = 0.5 * torch.ones(tt.arg.meta_batch_size, 2, num_supports + 1, num_supports + 1).to(tt.arg.device) # batch_size x 2 x (num_sup

    input_edge_feat[:, :, :num_supports, :num_supports] = init_edge[:, :, :num_supports, :num_supports] # batch_size x 2 x (num_support + 1) x (num
    input_edge_feat = input_edge_feat.repeat(num_queries, 1, 1, 1) #(batch_size x num_queries) x 2 x (num_support + 1) x (num_support + 1)

    # logit: (batch_size x num_queries) x 2 x (num_support + 1) x (num_support + 1)
    logit_layers = self.gnn_module(node_feat=input_node_feat, edge_feat=input_edge_feat)

    logit_layers = [logit_layer.view(tt.arg.meta_batch_size, num_queries, 2, num_supports + 1, num_supports + 1) for logit_layer in logit_layers]
```

EGNN 훈련 과정

- train.py
  - Compute loss

```
# (4) compute loss
full_edge_loss_layers = [self.edge_loss((1-full_logit_layer[:, 0]), (1-full_edge[:, 0])) for full_logit_layer in full_logit_layers]

# weighted edge loss for balancing pos/neg
pos_query_edge_loss_layers = [torch.sum(full_edge_loss_layer * query_edge_mask * full_edge[:, 0] * evaluation_mask) / torch.sum(query_edge_mask * full_edge[:, 0] * eval
neg_query_edge_loss_layers = [torch.sum(full_edge_loss_layer * query_edge_mask * (1-full_edge[:, 0]) * evaluation_mask) / torch.sum(query_edge_mask * (1-full_edge[:, 0]
query_edge_loss_layers = [pos_query_edge_loss_layer + neg_query_edge_loss_layer for (pos_query_edge_loss_layer, neg_query_edge_loss_layer) in zip(pos_query_edge_loss_la

# compute accuracy
full_edge_accr_layers = [self.hit(full_logit_layer, 1-full_edge[:, 0].long()) for full_logit_layer in full_logit_layers]
query_edge_accr_layers = [torch.sum(full_edge_accr_layer * query_edge_mask * evaluation_mask) / torch.sum(query_edge_mask * evaluation_mask) for full_edge_accr_layer in

# compute node loss & accuracy (num_tasks x num_quries x num_ways)
query_node_pred_layers = [torch.bmm(full_logit_layer[:, 0, num_supports:, :num_supports], self.one_hot_encode(tt.arg.num_ways_train, support_label.long())) for full_log
query_node_accr_layers = [torch.eq(torch.max(query_node_pred_layer, -1)[1], query_label.long()).float().mean() for query_node_pred_layer in query_node_pred_layers]

total_loss_layers = query_edge_loss_layers
```

EGNN 훈련 과정

- train.py
  - Model update

```
# update model
total_loss = []
for l in range(tt.arg.num_layers - 1):
    total_loss += [total_loss_layers[l].view(-1) * 0.5]
total_loss += [total_loss_layers[-1].view(-1) * 1.0]
total_loss = torch.mean(torch.cat(total_loss, 0))

total_loss.backward()

self.optimizer.step()

# adjust learning rate
self.adjust_learning_rate(optimizers=[self.optimizer],
                          lr=tt.arg.lr,
                          iter=self.global_step)

# logging
tt.log_scalar('train/edge_loss', query_edge_loss_layers[-1], self.global_step)
tt.log_scalar('train/edge_accr', query_edge_accr_layers[-1], self.global_step)
tt.log_scalar('train/node_accr', query_node_accr_layers[-1], self.global_step)
```

EGNN 훈련 과정

- train.py
  - Evaluation (for validation set)
  - Validation set 에 대해서 최고 성능 모델을 저장함
  - 일정 step지날 시에 best model을 계속 저장함

```
# evaluation
if self.global_step % tt.arg.test_interval == 0:
    val_acc = self.eval(partition='val')

    is_best = 0

    if val_acc >= self.val_acc:
        self.val_acc = val_acc
        is_best = 1

    tt.log_scalar('val/best_accr', self.val_acc, self.global_step)

    self.save_checkpoint({
        'iteration': self.global_step,
        'enc_module_state_dict': self.enc_module.state_dict(),
        'gnn_module_state_dict': self.gnn_module.state_dict(),
        'val_acc': val_acc,
        'optimizer': self.optimizer.state_dict(),
        }, is_best)

tt.log_step(global_step=self.global_step)
```

EGNN 실험 결과

- D-mini N-5 K-1 U-0 L-3 B-40 T-True 모델
  - Data : mini-imagenet N: 5-way K: 1-shot
  - U : unlabel 0%(fully-supervised)
  - L : number of node, edge network is 3
  - T : true = transductive setting
- Evaluation 결과 성능 59.18% (논문과 거의 유사)

```
(torch3) hyunjun@hyunjun:~/fewshot-egnn$ python3 eval.py --test_model D-mini_N-5_K-1_U-0_L-3_B-40_T-True
load pre-trained enc_nn done!
load pre-trained egnn done!
90000
/home/hyunjun/anaconda3/envs/torch3/lib/python3.6/site-packages/torch/nn/functional.py:1351: UserWarning: n
h.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.")
---------------------------
evaluation: total_count=999, accuracy: mean=59.18%, std=8.63%, ci95=0.53%
---------------------------
```

EGNN 코드 링크

- 업데이트 된 코드 링크 : https://github.com/hyunjunChhoi/fewshot-egnn

- 원본 코드 링크 : https://github.com/khy0809/fewshot-egnn

- Reference :
  - Kim, Jongmin, et al. "Edge-labeling graph neural network for few-shot learning." Proceed ings of the IEEE Conference on Computer Vision and Pattern Recognition. 2019.