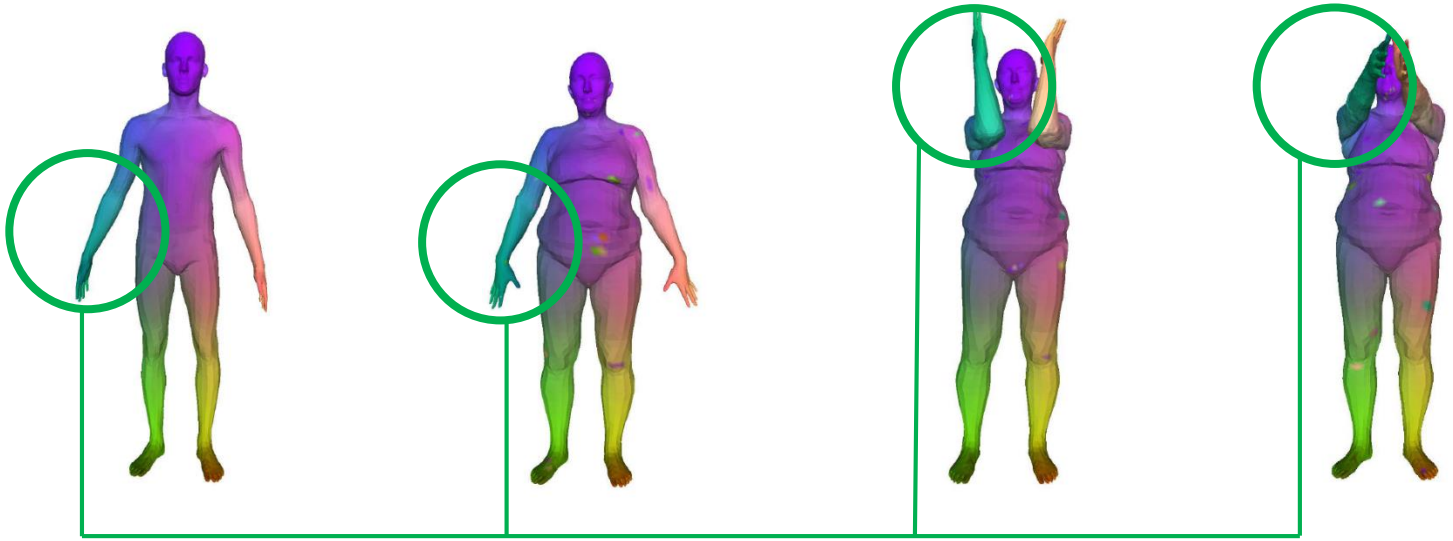
FeaStNet: Feature-Steered Graph Convolutions for 3D Shape Analysis

Tae Gyun Ahn

Seoul National University

Correspondence Learning



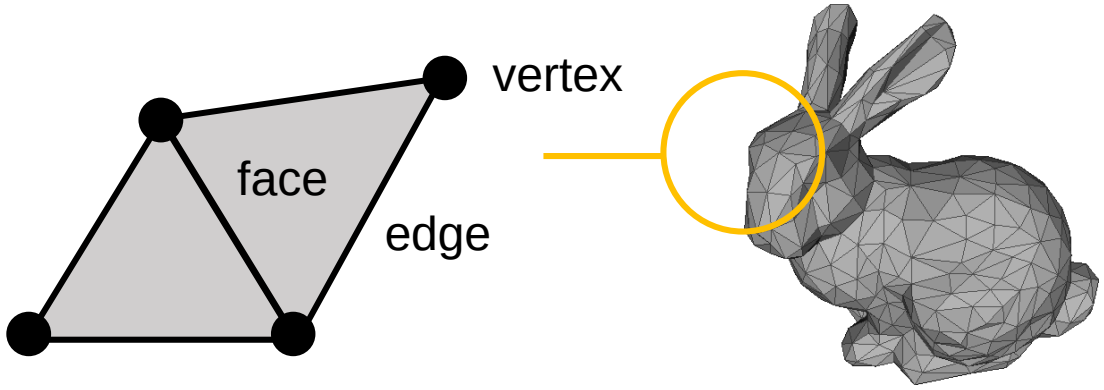
- given input shapes $S_1, S_2, \cdots S_N$,
  find a meaningful relation (mapping) between their elements

- requires an understanding of the structure of the shapes
  at both local and global levels

Motivation

- **Mesh**
  - 3D shape models can be represented using mesh
  - robust to many shape transformations
  - Describe 3D entities more efficiently with discretization that are attached to shapes
- **a graph structured** data:

  usually consists of vertices, edge, and face data
- Mesh is **irregular** structure: vertices can have a varying number of neighbors
- **CANNOT** use CNN
- **Need GCN!**



Contribution

1. **Dynamically** determine the **association** between **filer weights** and the **nodes**, using learned features of the preceding network layer

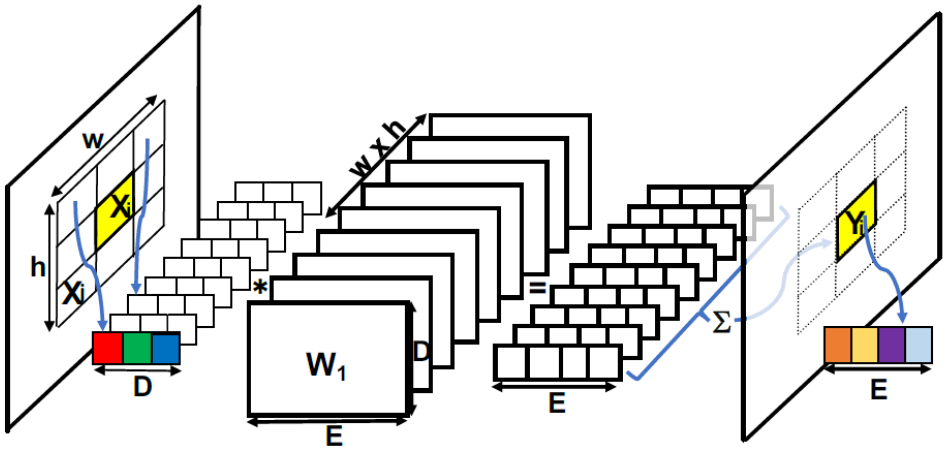
2. Can learn correspondences using **raw 3D shape coordinates** instead of 3D shape descriptors

3. Can be generalized to 3D data without explicit surface information

Related Works: Problems with previous GCN

- **1. Spectral Filtering**
  - Successful with synthetic 3D shape model (noise free data)
  - Not suitable for real shape models
  - Since global decompositions are unstable across different graphs
- **2. Local Filtering**
  - rely on sub optimal hard-coded methods using local pseudo coordinates to define filters

Method: CNN



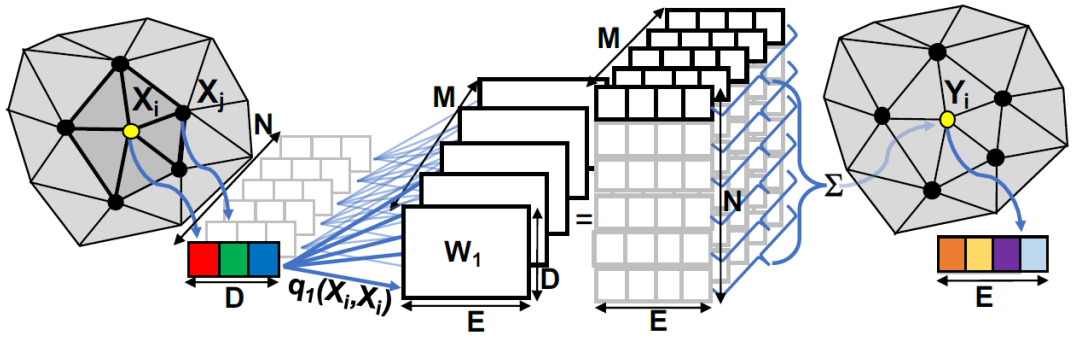
$x_{n(1,i)}$	$x_{n(2,i)}$	$x_{n(3,i)}$		W_1	W_2	W_3
$x_{n(4,i)}$	$x_{n(5,i)}$	$x_{n(6,i)}$	*	W_4	W_5	W_6
$x_{n(7,i)}$	$x_{n(8,i)}$	$x_{n(9,i)}$		W_7	W_8	W_9

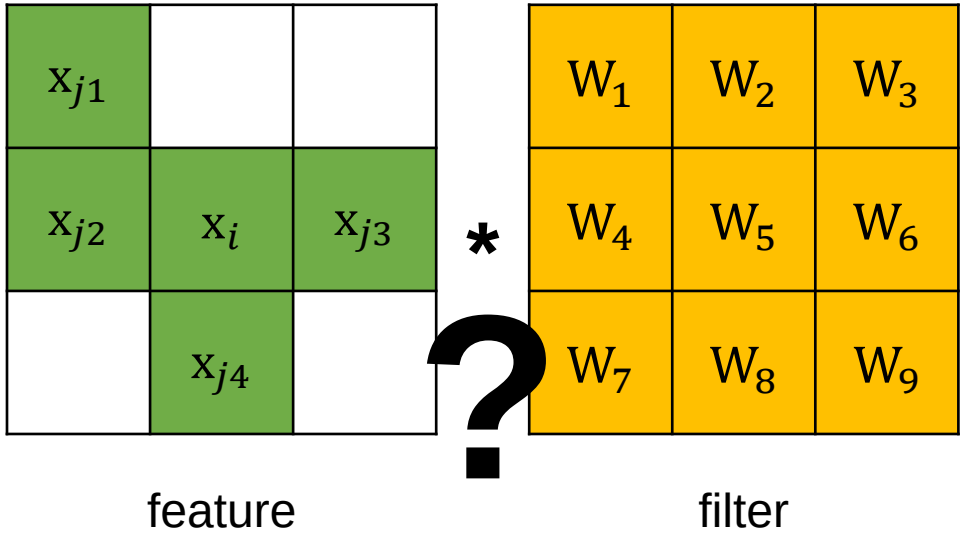
feature filter

$$y_i = b + \sum_{m=1}^{M} W_m x_{n(m,i)}$$

- $x_{n(m,i)} \in \mathbb{R}^D$, $y_i \in \mathbb{R}^E$, where D and E are number of channels
- $W_m \in \mathbb{R}^{E \times D}$: weight matrix of mth neighbor, $b \in \mathbb{R}^E$: bias
- $n(m,i)$: global index of mth neighbor

Method: GCN using node to weight association

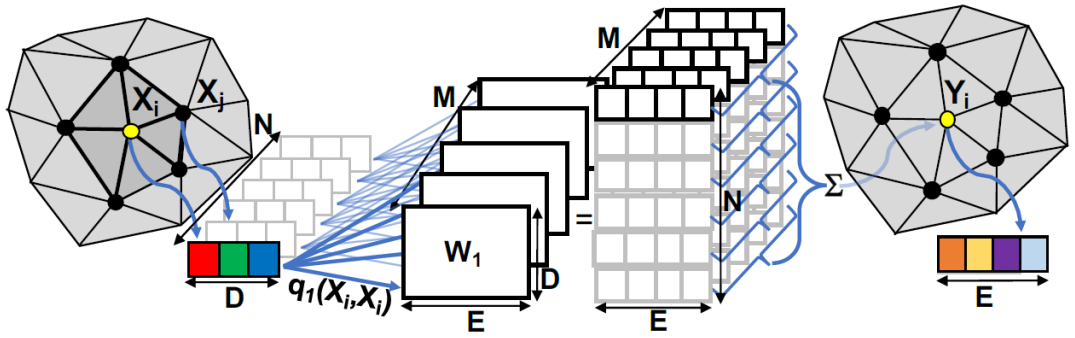




$$y_i = b + \sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(\mathrm{x}_i, \mathrm{x}_j) \mathrm{W}_m \mathrm{x}_j$$

- $q_m(\mathrm{x}_i, \mathrm{x}_j)$: **association** - assignment of x_j to W_m, $\sum_{\mathrm{m}=1}^{\mathrm{M}} q_m(\mathrm{x}_i, \mathrm{x}_j) = 1$

Method: GCN using node to weight association



x_{j1} *

$q_1(x_i, x_{j1})W_1$	$q_2(x_i, x_{j1})W_2$	$q_3(x_i, x_{j1})W_3$
$q_4(x_i, x_{j1})W_4$	$q_5(x_i, x_{j1})W_5$	$q_6(x_i, x_{j1})W_6$
$q_7(x_i, x_{j1})W_7$	$q_8(x_i, x_{j1}W_8$	$q_9(x_i, x_{j1})W_9$

feature filter

$$y_i = b + \sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(x_i, x_j) W_m x_j$$

- $q_m(x_i, x_j)$: **association** - assignment of x_j to W_m, $\sum_{m=1}^{M} q_m(x_i, x_j) = 1$

Method: GCN using node to weight association

$$y_i = b + \sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(\mathrm{x}_i, \mathrm{x}_j) \mathrm{W}_m \mathrm{x}_j$$

- $q_m(\mathrm{x}_i, \mathrm{x}_j) \propto \exp(\mathrm{u}_m^T \mathrm{x}_j + \mathrm{v}_m^T \mathrm{x}_i + c_m)$

  $\mathrm{u}_m, \mathrm{v}_m, c_m$: parameters of linear transformation.
- Translation invariant in feature space:

  $$q_m(\mathrm{x}_i, \mathrm{x}_j) \propto \exp(\mathrm{u}_m^T(\mathrm{x}_j - \mathrm{x}_i) + c_m)$$
- Robust to variations in the degree of the nodes:

  $$\sum_{j \in \mathcal{N}_i} \frac{1}{|\mathcal{N}_i|} \sum_{m=1}^{M} q_m(\mathrm{x}_i, \mathrm{x}_j) = 1$$
- $\mathcal{N}_i$ can be expended to higher degree neighbors

Comparison

CNN	GCN
$\mathrm{y}_i = b + \sum_{m=1}^{M} \mathrm{W}_m \mathrm{x}_{n(m,i)}$	$y_i = b + \sum_{m=1}^{M} \frac{1}{\lvert\mathcal{N}_i\rvert} \sum_{j \in \mathcal{N}_i} q_m(\mathrm{x}_i, \mathrm{x}_j) \mathrm{W}_m \mathrm{x}_j$
single node x_j - single weight matrix W_j	single node x_j - multiple weight matrix W_m $m = 1, \dots, \lvert\mathcal{N}_i\rvert$ (enough with 8)
Cost of computation: $\mathcal{O}(NMED)$ D : number of input channel	Cost of computation: $\mathcal{O}\big(NME(K+D)\big)$ K : average number of neighbors

Experiments

3D shape correspondence between 3D meshes using FAUST human shape dataset (dataset consists of 100 watertight meshes with 6,890 vertices each, corresponding to 10 shapes in 10 different poses each)

Architecture: Lin16+Conv32+Conv64+Conv128+Lin256+Lin6890 **3 layer of GCN**
Loss: cross-entropy classification loss

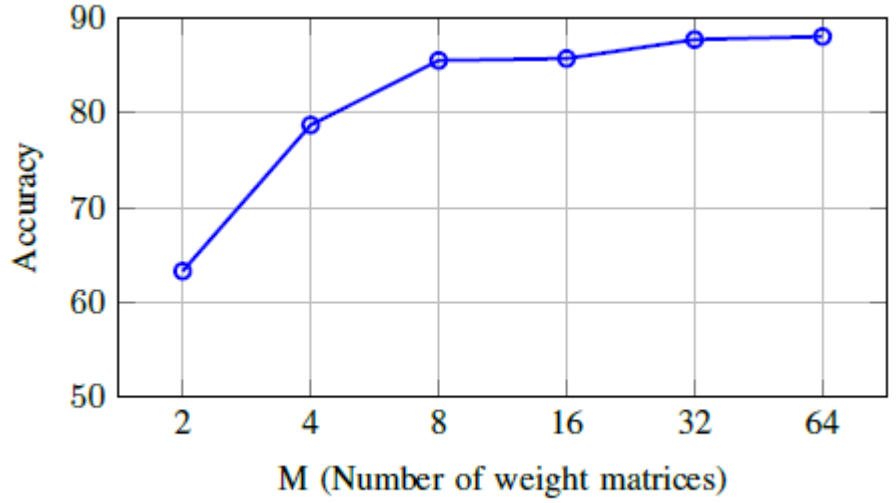
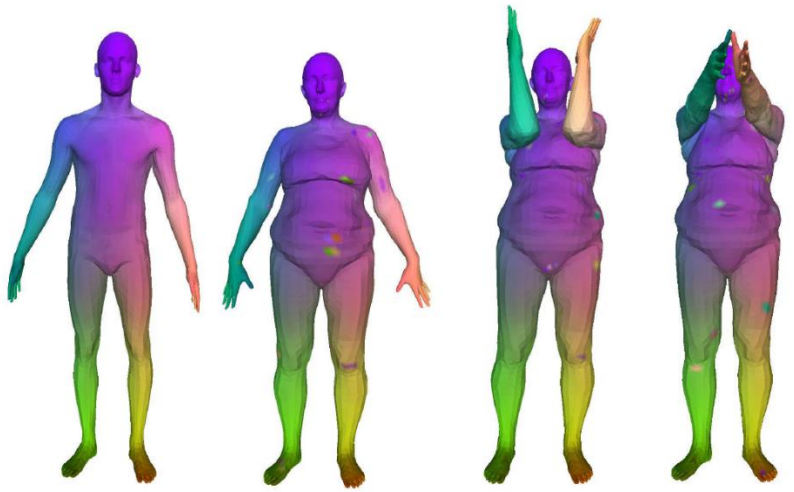


Figure 5. Accuracy as a function of the number of weight matrices



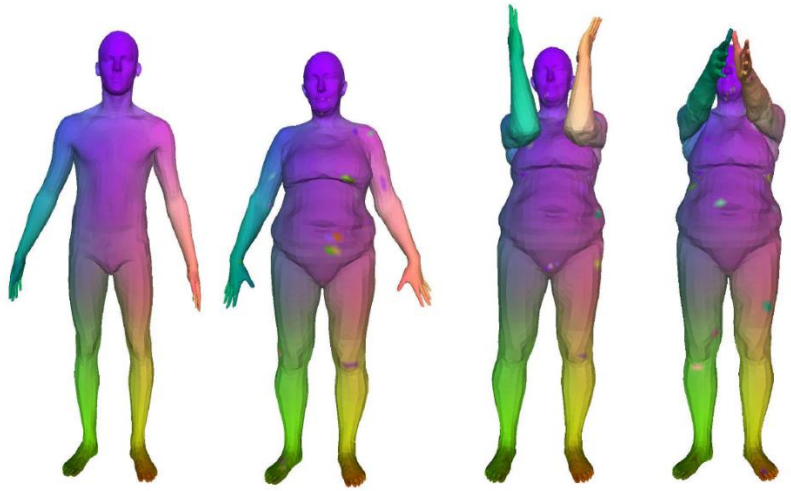
8 weight matrices are good enough!

Experiments

3D shape correspondence between 3D meshes using FAUST human shape dataset (dataset consists of 100 watertight meshes with 6,890 vertices each, corresponding to 10 shapes in 10 different poses each)

Architecture: Lin16+Conv32+Conv64+Conv128+Lin256+Lin6890 **3 layer of GCN**
Loss: cross-entropy classification loss

Method	Input	Accuracy
Logistic Regr.	SHOT	39.9%
PointNet [19]	SHOT	49.7%
GCNN [14], w/o refinement	SHOT	42.3%
GCNN [14], w/ refinement	SHOT	65.4%
ACNN [2], w/o refinement	SHOT	60.6%
ACNN [2], w/ refinement [17]	SHOT	62.4%
MoNet [15], w/o refinement	SHOT	73.8%
MoNet [15], w/ refinement [29]	SHOT	88.2%
FeaStNet, w/o refinement	XYZ	88.1%
FeaStNet, w/ refinement [29]	XYZ	92.2%
FeaStNet, multi scale, w/o refinement	XYZ	98.6%
FeaStNet, multi scale, w/ refinement [29]	XYZ	98.7%
FeaStNet, multi scale, w/o refinement	SHOT	90.9%

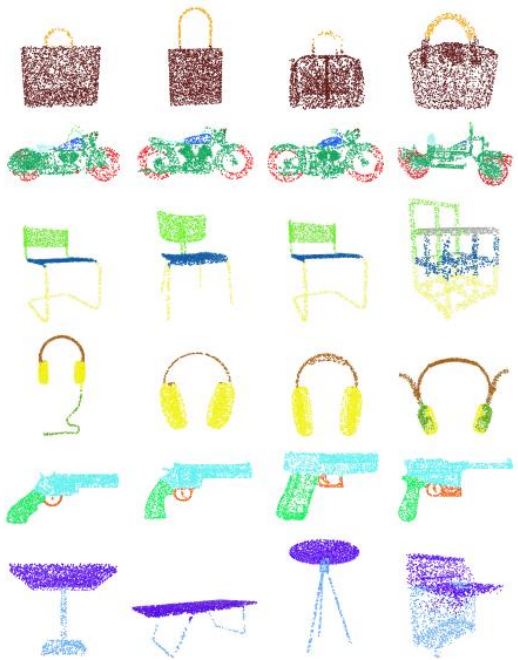


Does not need descriptors!

Experiments

3D Part labeling of point clouds from ShapeNet dataset
(16881 shapes from 16 categories, 50 labeled parts)
Points are **partially** labeled.
16-nearest neighbors were made to sub-graph to learn part labeling

Architecture: Lin16+**Conv32**+**Conv64**+**Conv128**+Lin512+Lin2048-MaxPool **3 layer of GCN**



	overall	aero plane	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor bike	mug	pistol	rocket	skate board	table
Number of shapes	16,881	2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
Wu [31]	-	63.2	-	-	-	73.5	-	-	-	74.4	-	-	-	-	-	-	74.8
Yi [33]	81.4	81.0	78.4	77.7	75.7	87.6	61.9	92.0	85.4	82.5	95.7	70.6	91.9	85.9	53.1	69.8	75.3
PointNet [19]	83.7	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6
Kd-network [12]	82.3	80.1	74.6	74.3	70.3	88.6	73.5	90.2	87.2	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3
FeaStNet (this paper)	81.5	79.3	74.2	69.9	71.7	87.5	64.2	90.0	80.1	78.7	94.7	62.4	91.8	78.3	48.1	71.6	79.6

Table 3. Part labeling accuracy in mIoU on the ShapeNet part dataset of our model and recent state-of-the-art approaches.

Conclusion

1. Introduced **association** between filer weights and the nodes for convolution of irregular graph structure with **finite** filter weights

2. Learned correspondences using **raw 3D shape coordinates** instead of 3D shape descriptors

Experiment Reproduction

- **3D shape correspondence** between 3D meshes using FAUST human shape dataset (slide 12)
- **Code Link**: https://github.com/sw-gong/FeaStNet (Pytorch)
- **Requirements**

  Pytorch (1.3.0) with GPU

  Pytorch Geoemtric (1.3.0)

  (but worked fine with Pytorch 1.4)
- This code is shared by the author of “SpiralNet++: A Fast and Highly Efficient Mesh Convolution Operator” (ICCV2019). He built this FeaStNet code to compare performance with his own method(spiralnet++).

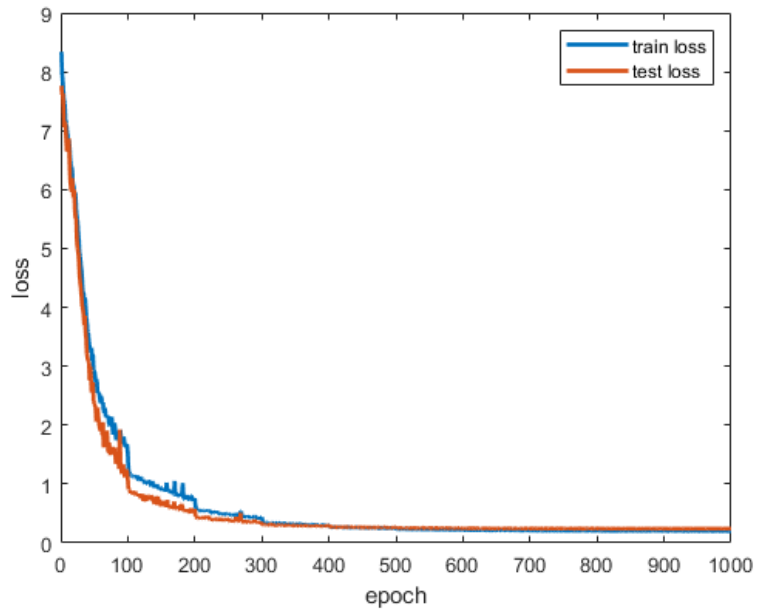
- Author of the original paper shared original source code based on TensorFlow, but the code lacks data loader.
- https://github.com/nitika-verma/FeaStNet (TensorFlow)

Experiment Reproduction

- **How to Run**
- 1. Clone git https://github.com/sw-gong/FeaStNet
- 2. Download FAUST data set from the homepage
  - http://faust.is.tue.mpg.de/
  - (we did not included the dataset to the git due to the size)
- 3. Make directory data/FAUST/raw in FeaStNet
- 4. Move MPI-FAUST.zip (uncompressed) to the above directory
- 5. run python main.py in FeaStNet

Reproduction Results

dataset consists of 100 watertight meshes with 6,890 vertices each
80 meshes were used for training and 20 meshes are used for testing
Dataset was trained for 1000 epochs



Accuracy: 93.4 %
This experiment corresponds to FeaStNet without refinement. Final accuracy was bit higher than the accuracy reported on paper

reproduction

Method	Input	Accuracy
Logistic Regr.	SHOT	39.9%
PointNet [19]	SHOT	49.7%
GCNN [14], w/o refinement	SHOT	42.3%
GCNN [14], w/ refinement	SHOT	65.4%
ACNN [2], w/o refinement	SHOT	60.6%
ACNN [2], w/ refinement [17]	SHOT	62.4%
MoNet [15], w/o refinement	SHOT	73.8%
MoNet [15], w/ refinement [29]	SHOT	88.2%
FeaStNet, w/o refinement	XYZ	88.1%
FeaStNet, w/ refinement [29]	XYZ	92.2%
FeaStNet, multi scale, w/o refinement	XYZ	98.6%
FeaStNet, multi scale, w/ refinement [29]	XYZ	98.7%
FeaStNet, multi scale, w/o refinement	SHOT	90.9%

original

Code Explanation

- About the data set
- Faust data set is given as .ply file which include ‘vertices’ with x,y,z c oordinates and ‘face’ with vertices index of the face
- The order of the vertices are corresponding points.
- Ex) if first vertex of human data1 and first vertex of human data2 are corresponding points
- Therefore, goal is to figure out the number of the vertex when x,y,z c oordinates and faces are given

Code Explanation

core part of main.py

```
41  class Pre_Transform(object):
42      def __call__(self, data):
43          data.x = data.pos
44          data = T.FaceToEdge()(data)
45          data.pos = None
46          return data
47
48
49  train_dataset = FAUST(args.data_fp, True, pre_transform=Pre_Transform())
50  test_dataset = FAUST(args.data_fp, False, pre_transform=Pre_Transform())
51  train_loader = DataLoader(train_dataset, batch_size=1, shuffle=True)
52  test_loader = DataLoader(test_dataset, batch_size=1)
53  d = train_dataset[0]
54  target = torch.arange(d.num_nodes, dtype=torch.long, device=device)
55  print(d)
56  model = FeaStNet(d.num_features, d.num_nodes, args.heads).to(device)
57  print(model)
58  optimizer = optim.Adam(model.parameters(),
59                         lr=args.lr,
60                         weight_decay=args.weight_decay)
61  scheduler = optim.lr_scheduler.StepLR(optimizer,
62                                        args.decay_step,
63                                        gamma=args.lr_decay)
64
65  run(model, train_loader, test_loader, target, d.num_nodes, args.epochs,
66      optimizer, scheduler, device)
```

.ply file of FAUST dataset does n ot have adjacency list. This part makes adjacency list of mesh ver tices using face vertices index inf ormation

1. Load train and test data
2. **Set network model to netw ork proposed in FeaStNet**
3. Set optimizer(Adam optimizer)
4. Execute training

Code Explanation

2. Setting Network (/models/networks.py)

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from .conv import FeaStConv


class FeaStNet(torch.nn.Module):
    def __init__(self, in_channels, num_classes, heads, t_inv=True):
        super(FeaStNet, self).__init__()

        self.fc0 = nn.Linear(in_channels, 16)
        self.conv1 = FeaStConv(16, 32, heads=heads, t_inv=t_inv)
        self.conv2 = FeaStConv(32, 64, heads=heads, t_inv=t_inv)
        self.conv3 = FeaStConv(64, 128, heads=heads, t_inv=t_inv)
        self.fc1 = nn.Linear(128, 256)
        self.fc2 = nn.Linear(256, num_classes)

        self.reset_parameters()

    def reset_parameters(self):
        self.conv1.reset_parameters()
        self.conv2.reset_parameters()
        self.conv3.reset_parameters()

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.elu(self.fc0(x))
        x = F.elu(self.conv1(x, edge_index))
        x = F.elu(self.conv2(x, edge_index))
        x = F.elu(self.conv3(x, edge_index))
        x = F.elu(self.fc1(x))
        x = F.dropout(x, training=self.training)
        x = self.fc2(x)
        return F.log_softmax(x, dim=1)
```

Set network architecture to the proposed architecture in slide 11:
Lin16+**Conv32**+**Conv64**+**Conv128**+Lin256+Lin6890

Code Explanation

2. Setting Network – Defining graph convolution (models/conv/feastconv.py)

```python
class FeaStConv(MessagePassing):
    def __init__(self,
                 in_channels,
                 out_channels,
                 heads=8,
                 bias=True,
                 t_inv=True,
                 **kwargs):
        super(FeaStConv, self).__init__(aggr='mean', **kwargs)

        self.in_channels = in_channels
        self.out_channels = out_channels
        self.heads = heads
        self.t_inv = t_inv

        self.weight = Parameter(torch.Tensor(in_channels,
                                             heads * out_channels))
        self.u = Parameter(torch.Tensor(in_channels, heads))
        self.c = Parameter(torch.Tensor(heads))
        if not self.t_inv:
            self.v = Parameter(torch.Tensor(in_channels, heads))

        if bias:
            self.bias = Parameter(torch.Tensor(out_channels))
        else:
            self.register_parameter('bias', None)

        self.reset_parameters()

    def reset_parameters(self):
        normal(self.weight, mean=0, std=0.1)
        normal(self.u, mean=0, std=0.1)
        normal(self.c, mean=0, std=0.1)
        normal(self.bias, mean=0, std=0.1)
        if not self.t_inv:
            normal(self.v, mean=0, std=0.1)
```

Setting weights for graph convolution:

$\mathbf{W_m}, \boldsymbol{u_m}, \boldsymbol{v_m}, \boldsymbol{c_m}, \boldsymbol{b}$

$$y_i = \boldsymbol{b} + \sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(\mathrm{x}_i, \mathrm{x}_j) \mathbf{W_m} \mathrm{x}_j$$

$$q_m(\mathrm{x}_i, \mathrm{x}_j) \propto \exp\left(\boldsymbol{u_m^T} \mathrm{x}_j + \boldsymbol{v_m^T} \mathrm{x}_i + \boldsymbol{c_m}\right)$$

*Translation invariant version:

$$q_m(\mathrm{x}_i, \mathrm{x}_j) \propto \exp(\boldsymbol{u_m^T}(\mathrm{x}_j - \mathrm{x}_i) + \boldsymbol{c_m})$$

Code Explanation

2. Setting Network – Defining graph convolution (models/conv/feastconv.py)

```
57    def message(self, x_i, x_j):
58        # dim: x_i, [E, F_in];
59        if self.t_inv:
60            # with translation invariance
61            q = torch.mm((x_i - x_j), self.u) + self.c  #[E, heads]
62        else:
63            q = torch.mm(x_i, self.u) + torch.mm(x_j, self.v) + self.c
64        q = F.softmax(q, dim=1)  #[E, heads]
65
66        x_j = torch.mm(x_j, self.weight).view(-1, self.heads,
67                                              self.out_channels)
68        return (x_j * q.view(-1, self.heads, 1)).sum(dim=1)
69
70    def update(self, aggr_out):
71        if self.bias is not None:
72            aggr_out = aggr_out + self.bias
73        return aggr_out
```

- 1. Setting association: $q_m(\mathrm{x}_i, \mathrm{x}_j)$
  - Translation invariant version:
    $q_m(\mathrm{x}_i, \mathrm{x}_j) \propto \exp(\boldsymbol{u}_m^T(\mathrm{x}_j - \mathrm{x}_i) + \boldsymbol{c}_m)$
  - General version:
    $q_m(\mathrm{x}_i, \mathrm{x}_j) \propto \exp(\boldsymbol{u}_m^T \mathrm{x}_j + \boldsymbol{v}_m^T \mathrm{x}_i + \boldsymbol{c}_m)$
- 2. Convolution:
  $$\sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(\mathrm{x}_i, \mathrm{x}_j) \mathrm{W}_m \mathrm{x}_j$$
  - Add bias:
  $$y_i = b + \sum_{m=1}^{M} \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} q_m(\mathrm{x}_i, \mathrm{x}_j) \mathrm{W}_m \mathrm{x}_j$$

Code Explanation

4. Execute training (line 65 of main.py) (utils/train_eval.py)

```
14  def run(model, train_loader, test_loader, target, num_nodes, epochs, optimizer,
15          scheduler, device):
16
17      for epoch in range(1, epochs + 1):
18          t = time.time()
19          train_loss = train(model, train_loader, target, optimizer, device)
20          t_duration = time.time() - t
21          scheduler.step()
22          acc, test_loss = test(model, test_loader, num_nodes, target, device)
23          eval_info = {
24              'train_loss': train_loss,
25              'test_loss': test_loss,
26              'acc': acc,
27              'current_epoch': epoch,
28              'epochs': epochs,
29              't_duration': t_duration
30          }
31
32          print_info(eval_info)
```

- train
- test

Code Explanation

4. Execute training (line 65 of main.py) (utils/train_eval.py)

```
14  def run(model, train_loader, test_loader, target, num_nodes, epochs, optimizer,
15          scheduler, device):
16
17      for epoch in range(1, epochs + 1):
18          t = time.time()
19          train_loss = train(model, train_loader, target, optimizer, device)
20          t_duration = time.time() - t
21          scheduler.step()
22          acc, test_loss = test(model, test_loader, num_nodes, target, device)
23          eval_info = {
24              'train_loss': train_loss,
25              'test_loss': test_loss,
26              'acc': acc,
27              'current_epoch': epoch,
28              'epochs': epochs,
29              't_duration': t_duration
30          }
31
32          print_info(eval_info)
```

- train
- test

Code Explanation

4. Execute training (utils/train_eval.py)

```
35 def train(model, train_loader, target, optimizer, device):
36     model.train()
37
38     total_loss = 0
39     for idx, data in enumerate(train_loader):
40         optimizer.zero_grad()
41         loss = F.nll_loss(model(data.to(device)), target)
42         loss.backward()
43         optimizer.step()
44         total_loss += loss.item()
45     return total_loss / len(train_loader)
46
47
48 def test(model, test_loader, num_nodes, target, device):
49     model.eval()
50     correct = 0
51     total_loss = 0
52     n_graphs = 0
53     with torch.no_grad():
54         for idx, data in enumerate(test_loader):
55             out = model(data.to(device))
56             total_loss += F.nll_loss(out, target).item()
57             pred = out.max(1)[1]
58             correct += pred.eq(target).sum().item()
59             n_graphs += data.num_graphs
60     return correct / (n_graphs * num_nodes), total_loss / len(test_loader)
```

- train

negative log likelihood loss

- test

Prediction is made by feeding test data to the model extracting feature with highest value

Accuracy is calculated:
number of correct prediction divided by number of vertices