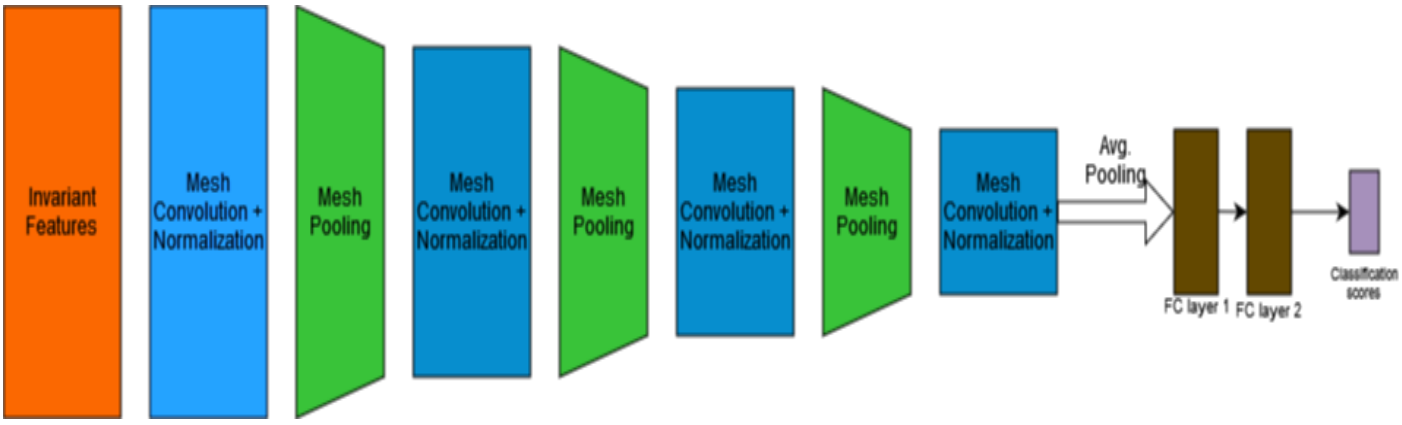
Project 4 - MeshCNN

Vamsi Krishna Satyanarayana Vasa - 1229524844
Aaron Saju Augustine - 1229572047

Introduction

MeshCNN is a Convolutional Neural Network that focuses on selecting edge-based features to represent meshes. The unique selection of features combined with their novel Mesh Pooling technique that helps simplify the mesh via mesh collapse in a way specific to the context of the task at hand, makes it easy for classifiers to label the mesh and perform other downstream tasks. Our project focuses on a set of experiments examining the effects of the various core components of MeshCNN.

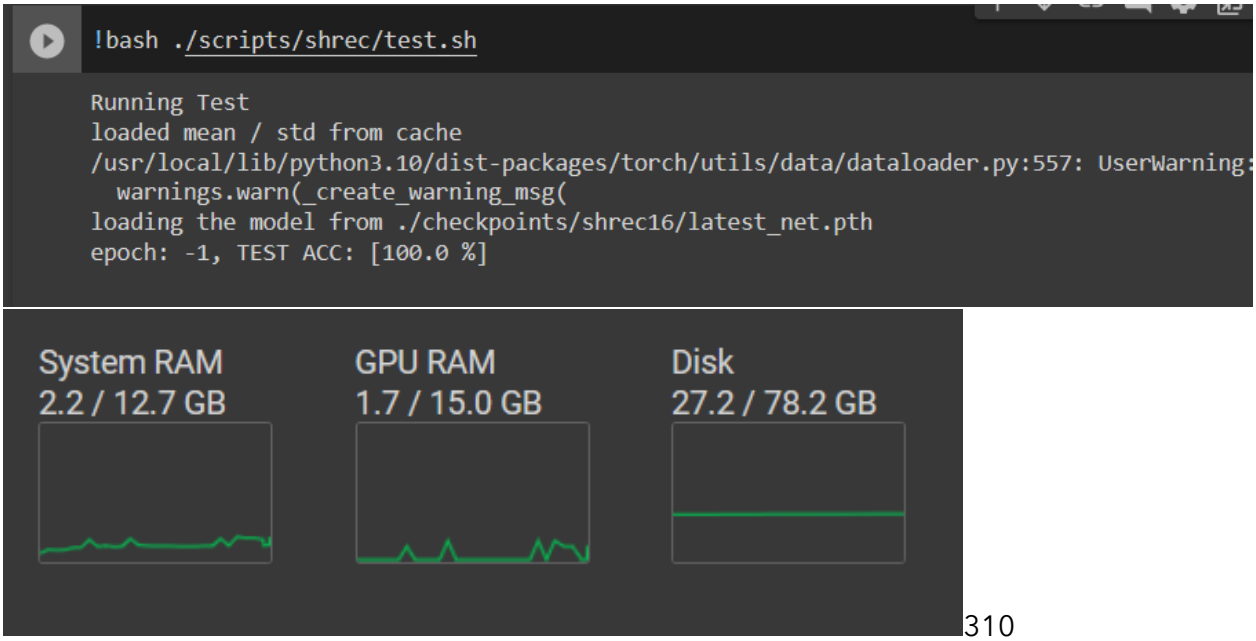
The rough architecture looks like this:



Experiment 1 – Original MeshCNN

Before starting out, we must ensure that training always starts from scratch and we must edit the options/train_options.py file to achieve this. Then, running the original code on our test dataset we got the following results:

Method	Num of Parameters	GPU Memory for Training	Test Accuracy
Original MeshCNN	1.321 million	1.7GB VRAM	100%



310

Experiment 2 – Altering the Edge Features (Average of Centroid Coordinates)

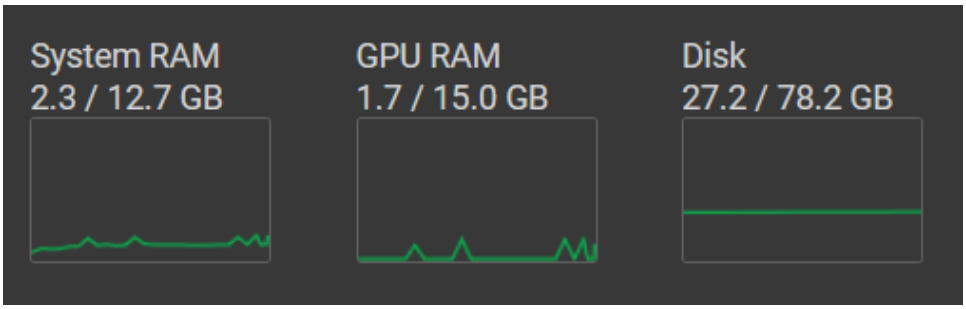
We change up the function extract_features on line 310 in mesh_prepare.py within the model/layers/ folder to extract the average of the 5 edge centroid coordinates for every edge and return the array as the set of features to be used:

```python

def extract_features(mesh):
    features = []
    edge_points = get_edge_points(mesh)
    """To consider all five edge centroids"""
    for i in range(edge_points.shape[0]):
      centroids = []
      edge_vertex_1 = np.array(mesh.vs[edge_points[i,0]])
      edge_vertex_2 = np.array(mesh.vs[edge_points[i,1]])
      edge_vertex_3 = np.array(mesh.vs[edge_points[i,2]])
      edge_vertex_4 = np.array(mesh.vs[edge_points[i,3]])

      centroid_vertex_1 = np.array(edge_vertex_1 + edge_vertex_2) / 2
      centroids.append(np.average(centroid_vertex_1))
      # centroids.append(np.average(centroid_vertex_1))
      centroid_vertex_2 = np.array(edge_vertex_1 + edge_vertex_3) / 2
      centroids.append(np.average(centroid_vertex_2))
      # centroids.append(np.average(centroid_vertex_2))
      centroid_vertex_3 = np.array(edge_vertex_1 + edge_vertex_4) / 2
      centroids.append(np.average(centroid_vertex_3))
      # centroids.append(np.average(centroid_vertex_3))
      centroid_vertex_4 = np.array(edge_vertex_2 + edge_vertex_4) / 2
      centroids.append(np.average(centroid_vertex_4))
      # centroids.append(np.average(centroid_vertex_4))
      centroid_vertex_5 = np.array(edge_vertex_2 + edge_vertex_3) / 2
      centroids.append(np.average(centroid_vertex_5))
      # centroids.append(np.average(centroid_vertex_5))
      centroids = np.array(centroids)
      features.append(centroids)
    features = np.array(features)
    # print(features.shape)
    return features.T

```

The results are as follows:

Method	Num of Parameters	GPU Memory for Training	Test Accuracy
Edge Centroid Average Coordinate	1.321 million	1.7GB VRAM	100%

```
!bash ./scripts/shrec/test.sh

Running Test
loaded mean / std from cache
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557:
  warnings.warn(_create_warning_msg(
loading the model from ./checkpoints/shrec16/latest_net.pth
epoch: -1, TEST ACC: [100.0 %]
```



Experiment 3 – Removed Mesh Pooling

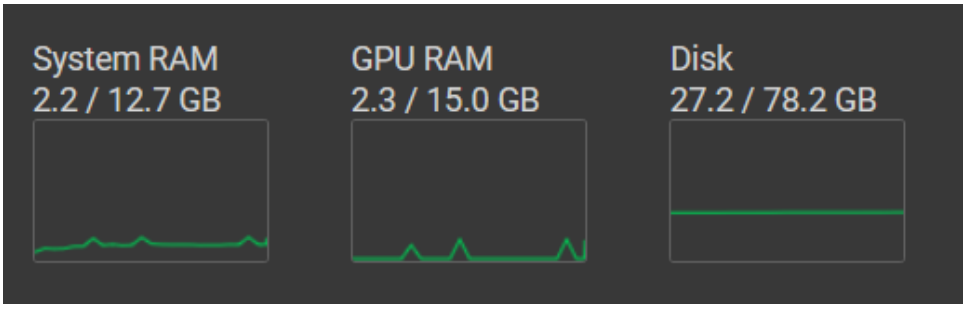
We change up the class MeshConvNet on line 125 of networks.py in the models/ folder to remove Mesh Pooling from the class structure as well as the forward pass function and rewrite Average Pooling to accept the final 750 edges as input

```
124
125 class MeshConvNet(nn.Module):
126     """Network for learning a global shape descriptor (classification)
127     """
128     def __init__(self, norm_layer, nf0, conv_res, nclasses, input_res, pool_res, fc_n,
129                  nresblocks=3):
130         super(MeshConvNet, self).__init__()
131         self.k = [nf0] + conv_res
132         self.res = [input_res] + pool_res
133         norm_args = get_norm_args(norm_layer, self.k[1:])
134 
135         for i, ki in enumerate(self.k[:-1]):
136             setattr(self, 'conv{}'.format(i), MResConv(ki, self.k[i + 1], nresblocks))
137             setattr(self, 'norm{}'.format(i), norm_layer(**norm_args[i]))
138             # setattr(self, 'pool{}'.format(i), MeshPool(self.res[i + 1]))
139 
140 
141         self.gp = torch.nn.AvgPool1d(750)
142         print(self.res)
143         # self.gp = torch.nn.AvgPool1d(self.res[-1])
144         # self.gp = torch.nn.MaxPool1d(self.res[-1])
145         # print("HE")
146         # print(self.k)
147         # print(fc_n)
148         self.fc1 = nn.Linear(self.k[-1], fc_n)
149         self.fc2 = nn.Linear(fc_n, nclasses)
150 
151     def forward(self, x, mesh):
152 
153         for i in range(len(self.k) - 1):
154             x = getattr(self, 'conv{}'.format(i))(x, mesh)
155             x = F.relu(getattr(self, 'norm{}'.format(i))(x))
156             # x = getattr(self, 'pool{}'.format(i))(x, mesh)
157 
158         print(x.shape)
159         x = x.squeeze()
160         print(x.shape)
161         x = self.gp(x)
162         x = x.view(-1, self.k[-1])
163 
164         x = F.relu(self.fc1(x))
165         x = self.fc2(x)
166         return x
167 
```

The results are as follows:

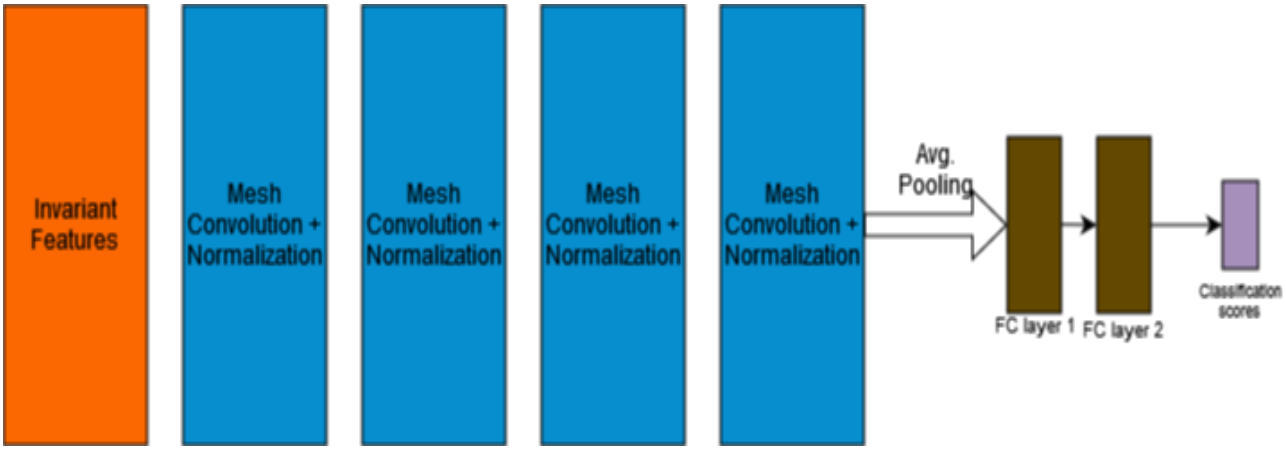
Method	Num of Parameters	GPU Memory for Training	Test Accuracy
Removed Mesh Pooling	1.321 million	2.3GB VRAM	100%

```
Running Test
loaded mean / std from cache
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserW
  warnings.warn(_create_warning_msg(
[750, 600, 450, 300, 180]
HE
[3, 64, 128, 256, 256]
100
loading the model from ./checkpoints/shrec16/latest_net.pth
torch.Size([8, 256, 750, 1])
torch.Size([8, 256, 750])
epoch: -1, TEST ACC: [100.0 %]
```



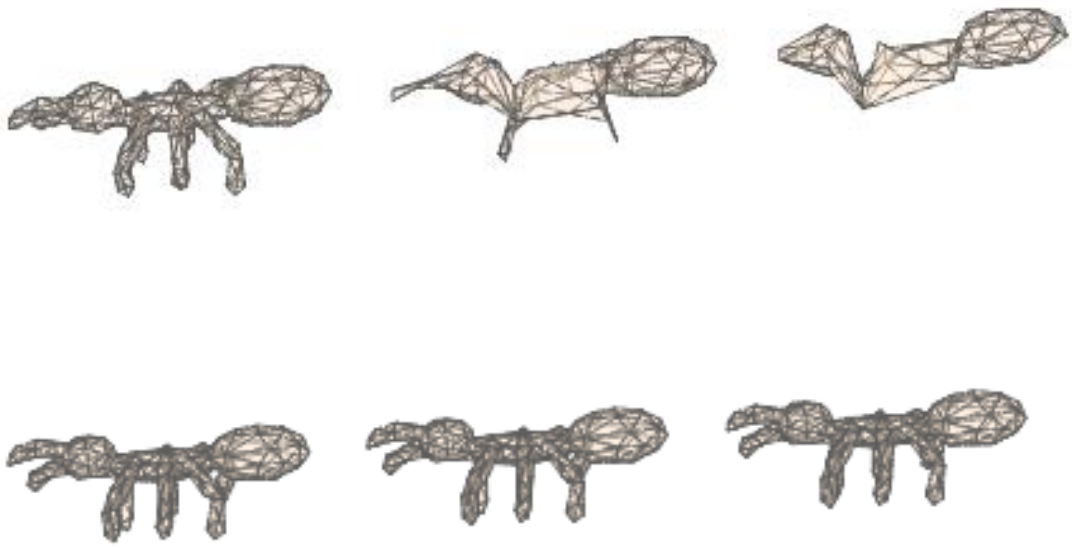
Additional Observations

The newly altered network architecture after Experiment 3 looks like this:



Here we can see that due to a lack of Mesh Pooling, the complexity of the mesh features and number of edges do not drop after every iteration of convolution. The expected drop from 750 to 180 features does not happen. Thus, we retain more features within memory requiring more VRAM to train the network and preventing the simplification of the mesh within the context of the task, leading to a possible drop in accuracy on larger datasets.

The results on the mesh of an Ant model with Mesh Pooling and without Mesh Pooling respectively are indicated below:



For Experiment 2, we lose the edge-specific features making use of their geodesic measurements and instead rely on coordinates which carry less information. This will also impact the accuracy on larger datasets.

Conclusion

All experiments were run for 50 epochs, and we were able to examine the effects of the various components on the architecture and inference quality of MeshCNN