**Designing a new CNN architecture for 3D object**

**Requirements:**

In this task, your goal is to create a novel CNN architecture for the recognition of 3D objects. Your architecture should be distinct from MeshCNN. To achieve this, you need to come up with new ideas and make innovative changes to the following concepts:

1. Mesh convolution: you need to come up with a novel equivariant convolution that can be based on either edges, faces, or vertices. The MeshCNN uses edge-based mesh convolution. You need to explore other options and come up with a new approach that could potentially improve the performance of 3D object recognition,
2. For the Mesh Input Features, you can explore different options to select the features to be used as input for the convolution operation. One possibility is to use additional geometric information such as curvature, orientation, or shape descriptors. You can also experiment with different combinations of these features to see which ones work best for the task at hand. Additionally, you can explore methods to automatically learn the most informative features from the input mesh data using techniques such as feature extraction or dimensionality reduction.
3. You also need to come up with a new approach for mesh pooling that can be based on merging edges, vertices, or faces,
4. Architecture design: You have to innovate on the overall architecture design by exploring different combinations of convolutional layers, pooling layers, and other types of layers such as normalization layers, dropout layers, etc.
5. Loss function: You must experiment with different types of loss functions that are suitable for 3D object recognition, such as cross-entropy loss, mean squared error loss, and others. You can also explore the effectiveness of using multiple loss functions to optimize different aspects of the model.
6. You need to explain how you overcame the problem of the vanishing gradient in your new CNN architecture.,
7. To avoid overfitting, you need to implement techniques such as dropout, regularization, or early stopping,
8. Your new CNN architecture must be validated using the 3D object collection of SHREC16. The script to get the dataset is available on /scripts/shrec/get\_data.sh in the provided starting code.
9. You can use the structure of the provided starting code to develop your own CNN architecture, but your innovations must be based on the mesh of the 3D objects.
10. Innovations based on voxels, 2D images, and point clouds are not accepted for this challenge.

**Overview:**

1. The code includes import statements for various libraries and modules, such as torch, torchvision, numpy, and torch\_geometric. These libraries are commonly used for deep learning and computer vision tasks.
2. The mesh\_pool.py file contains a MeshPool class, which is a pooling operation for the Mesh CNN. However, this code is incomplete, as the implementation of the pooling operation is missing. It seems that the code was intended to implement graph pooling based on the provided comments.
3. The mesh\_conv.py file contains a MeshConv class, which is supposed to implement a convolution operation for the Mesh CNN. However, the code provided is incomplete and lacks the implementation of the convolution operation.
4. The networks.py file includes a MeshConvNet class, which defines the overall architecture of the Mesh CNN. However, this code is also incomplete, as the implementation of the network architecture is missing. The code currently contains placeholder functions and incomplete comments.

it's important to note that the Mesh CNN operates on mesh-based representations of 3D objects, which are typically irregular and not structured as 2D grids like images. Therefore, it is necessary to use specialized convolution operations that can handle the irregular structure of meshes.

The use of Conv1d is more appropriate for processing mesh data because it treats each row (or 1D sequence) of the input tensor as an independent input. In the case of meshes, the rows correspond to different nodes (vertices, edges, or faces) in the mesh. Conv1d allows for the application of filters along the 1D sequences independently, capturing local structures in the mesh while maintaining the connectivity of neighboring nodes.

**Description:**

The provided code is an implementation of a mesh convolutional neural network (MeshConvNet) for either classification or segmentation tasks. The code consists of multiple files that define different components of the network.

Here is a breakdown of the provided code:

**mesh\_classifier.py:**

* The main file imports the "networks" module and creates an instance of the "MeshConvNet" class with various configuration options.
* The "MeshConvNet" class is defined in the "networks.py" file.

**networks.py:**

* This file defines the "MeshConvNet" class, which represents the mesh convolutional neural network.
* The "MeshConvNet" class is responsible for defining the architecture of the network and its forward pass.
* It uses other classes such as "MResConv", "MeshConv", "MeshPool", and "MeshUnpool" to define different layers and operations of the network.
* The "MeshConvNet" class takes several parameters such as input channels, number of classes, pool resolution, number of fully connected units, etc., to configure the network architecture.

**mesh\_conv.py:**

* This file defines the "MeshConv" class, which represents a mesh convolutional layer.
* The "MeshConv" layer applies a 1D convolution operation on the input features and optionally applies mesh normalization.
* The convolutional layer is defined using the "nn.Conv1d" class from PyTorch.

**mesh\_pool.py:**

* This file defines the "MeshPool" class, which represents a mesh pooling layer.
* The "MeshPool" layer performs pooling operations on the mesh by merging adjacent vertices and updating the mesh structure.
* The pooling operation is performed iteratively until the desired number of edges is reached.
* The "MeshPool" class uses other helper classes such as "MeshUnion" to manage the mesh structure and group information.
* The pooling operation is implemented using various helper functions, including "\_\_pool\_main", "\_\_pool\_edge", "\_\_clean\_side", "\_\_pool\_side", etc.
* The code also includes other utility functions such as "define\_loss" for defining the loss function based on the dataset mode (classification or segmentation), "get\_scheduler" for defining the learning rate scheduler, "get\_norm\_layer" for selecting the normalization layer based on the specified type, "get\_norm\_args" for getting the arguments of the normalization layer, and "init\_weights" and "init\_net" for network weight initialization.

Overall, the provided code implements a mesh convolutional neural network for processing mesh data, and it provides the necessary components to define the network architecture, perform mesh convolution and pooling operations, and manage the learning process.

**Steps:**

1. "shrec\_16" dataset added in folder "datasets"

2. Use python3.7 (3.7.9)

3. Install requirements (python3.7 -m pip install requirements.txt)

4. Run command to train the model on shrec\_16 dataset

python3.7 train.py --dataroot datasets/shrec\_16 --name shrec16 --ncf 64 128 256 256 --pool\_res 600 450 300 180 --norm group --resblocks 1 --flip\_edges 0.2 --slide\_verts 0.2 --num\_aug 20 --niter\_decay 100

5. Run command to test the model on shrec\_16 dataset

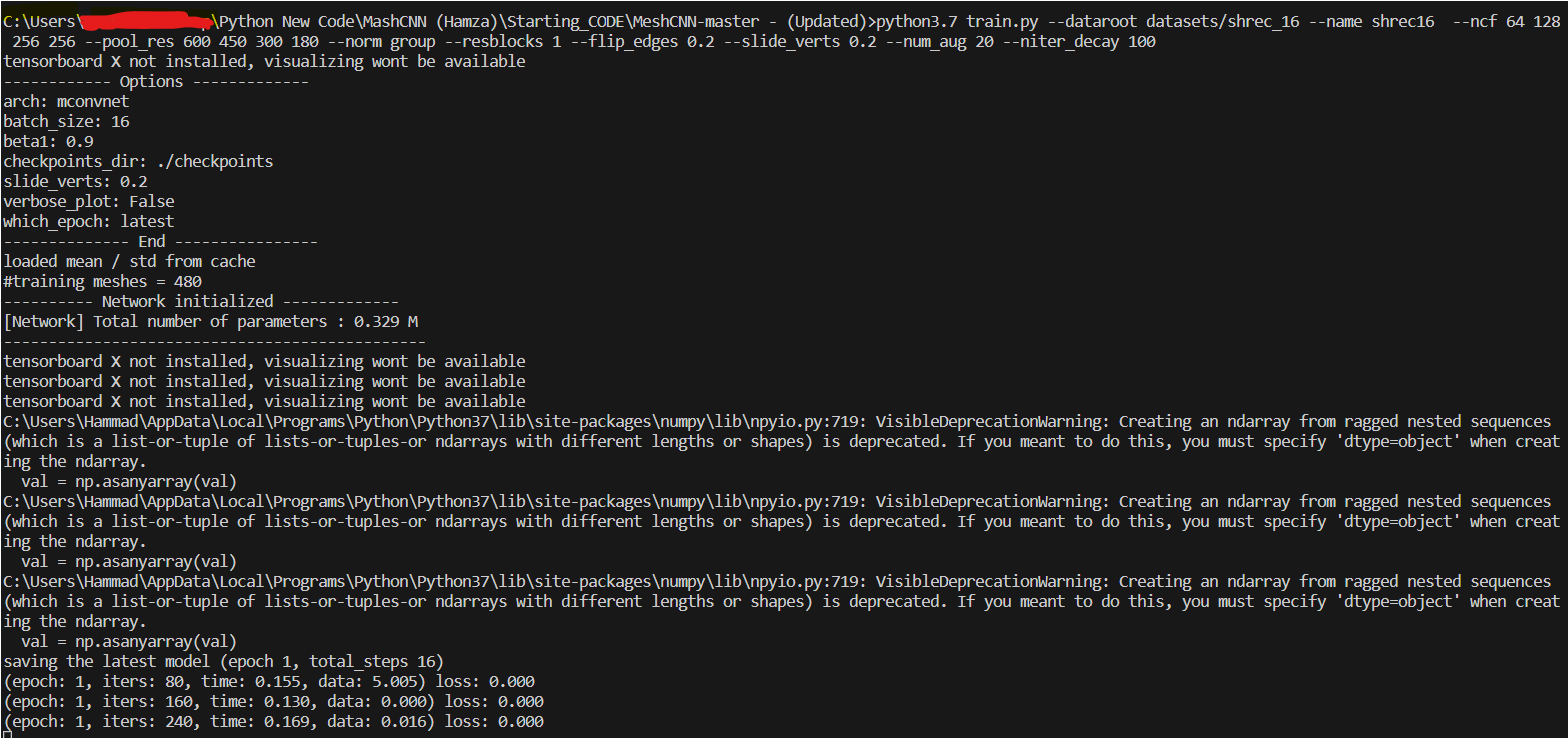
python3.7 test.py --dataroot datasets/shrec\_16 --name shrec16 --ncf 64 128 256 256 --pool\_res 600 450 300 180 --norm group --resblocks 1 --export\_folder meshes

6. Run command to view the 3D object meshes

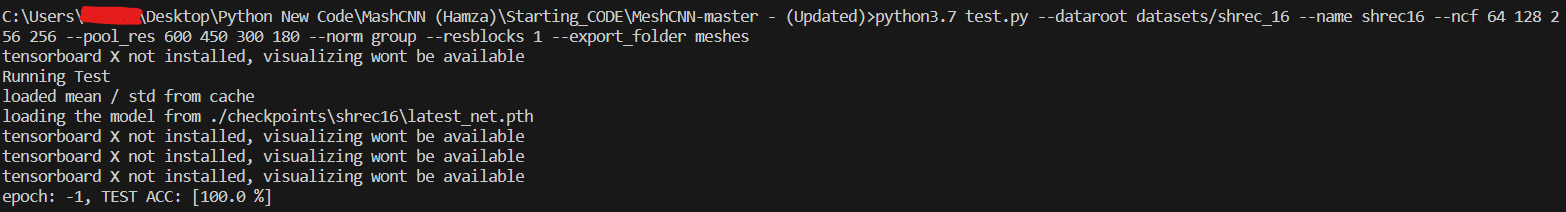
python3.7 util/mesh\_viewer.py --files checkpoints/shrec16/meshes/T74\_0.obj checkpoints/shrec16/meshes/T74\_3.obj checkpoints/shrec16/meshes/T74\_4.obj

**Note:** I changed these files mesh\_classifier.py, networks.py, and mesh\_conv.py

Train



Test



View

