Bishop's University

CS 590 – Master's Project

Designing a new CNN architecture for 3D object recognition

1. Introduction

The objective of this project is to develop a new CNN architecture for 3D object recognition. Utilizing machine learning on 3D data is more complex than on 2D data, as there are various representations for 3D data, and the selection of a representation determines the learning strategies that can be employed. To achieve this goal, this document is organized as follows: In Section 2, the various representations of 3D objects are discussed. To guide you in designing a new CNN architecture for 3D object recognition, instructions for this challenge are provided in Section 3.

2. Representations of 3D objects

Determining the most suitable way to represent 3D meshes for deep learning can be challenging. Unlike 2D RGB images, there is no consensus on the optimal representation. This is because the choice of representation influences the learning approach that needs to be taken. For instance, in classification tasks, the model can be projected from 3D space into 2D images, allowing the application of standard 2D convolutions. Alternatively, the 3D space occupied by the model can be represented as a grid of voxels, enabling the use of 3D convolutions. Another approach involves sampling the mesh's vertices as a 3D point cloud and applying specialized methods such as PointNet++ or 3D point capsule networks. There are also techniques like PolyGen, which directly processes the model's vertices and faces.

2.1 3D object projection into 2D images

In some of the earliest 3D deep learning studies, the issue of 3D representation was tackled by directly projecting the 3D model into 2D images. This approach makes 3D data compatible with the conventional vision CNN methods. For example, the Multi-view Convolutional Neural Networks for 3D Shape Recognition method described in a research paper projects a model into twelve unique viewpoints and pools their activations

to produce a softmax score. The paper reports a classification accuracy of 90.1% on the ModelNet40 dataset. As of today, the best performance on the ModelNet40 dataset according to the project's website is 97.37%, achieved by the RotationNet: Joint Object Categorization and Pose Estimation Using Multiviews from Unsupervised Viewpoints paper. Like the previous paper, this approach trains on multiple views to predict an object category but simultaneously predicts the viewpoint as well, treating it as a latent variable. It can accurately predict both object class and viewpoint, even for real-world objects.

However, although simple and elegant, the projection representation has some limitations. Firstly, it doesn't consider the complete topology of the 3D object. Secondly, it makes assumptions about how the model should be viewed. Thirdly, it doesn't provide a straightforward solution for non-global tasks like segmentation.

2.2 Volumetric methods

The volumetric methods also called voxel methods, provide a solution to the limitations of the projection method while still allowing the use of convolutions. To apply these methods, a dense occupancy grid is created by dividing the 3D space into bins and assigning a boolean value to each cell based on whether it falls inside the object. The grid structure is compatible with 3D convolution. One of the earliest papers to employ this approach is 3D ShapeNets: A Deep Representation for Volumetric Shapes, which also introduces the ModelNet40 dataset. The authors of this study aim to reconstruct the 3D structure of an object by predicting 3D voxels from a single depth image and achieve impressive results.

However, classification using 3D voxels presents some challenges. While satisfactory results can be obtained, the downside is that the large amount of empty space results in excessive convolutions. Moreover, the number of weights increases cubically with the model resolution, making it impractical to work with sizes greater than 256x256x256 in most cases, even when the sparsity of the space is exploited, as demonstrated in OctNet: Learning Deep 3D Representations at High Resolutions, which achieved approximately 86% accuracy on the ModelNet40 dataset while also improving speed and memory consumption. While high-resolution voxel spaces are expensive, low-resolution voxel spaces compromise the fine details of model topology that may be useful.

.

This article discusses a unique approach called MeshCNN: A Network with an Edge, which presents a generic framework for processing 3D models for classification and segmentation tasks. One of the most intriguing features of this approach is its mesh pooling operation, which enables the combination of mesh features at different scales, akin to a convolutional neural network. The mesh pooling operation is a learned process that gradually reduces a model to its most informative edges for a given task. MeshCNN integrates several beneficial properties of different 3D representations. To better understand these properties, let's briefly examine some of the commonly used 3D representations.

2.3 Point cloud

There are various techniques that directly process vector point clouds. An example of this is the 2019 paper titled Spherical Kernel for Efficient Graph Convolution on 3D Point Clouds, in which the authors were able to achieve an 89.3% accuracy on the ModelNet40 task using their method. One advantage of using 3D point cloud representation is its flexibility, since it can represent a range of objects from LiDAR scans to authored 3D models. Even popular models like PointNet and PointNet++ perform reasonably well on classification tasks (achieving 88.0% accuracy in the paper mentioned earlier). However, the downside of point cloud methods is that they typically do not consider surfaces. Without faces, the mesh's actual topology cannot be determined since a set of vectors does not provide a unique set of faces.

A high-performing technique on the ModelNet40 dataset is the Relation-Shape Convolutional Neural Network for Point Cloud Analysis, which achieves an accuracy of 93.6%. This approach, known as RS-CNN, employs geometric priors to deduce the underlying topology of the point cloud, giving the model spatial knowledge of the input points. This technique can be used for both point clouds and meshes, but it lacks a method for incorporating mesh data, even when it is accessible.

2.4 Mesh-based representation

The representation of 3D shapes using polygonal meshes is efficient since they capture the surface and topology of the object, including large flat regions and intricate details. However, the irregularity of meshes poses challenges for convolutional and pooling operations in neural networks for mesh analysis. The aim is to exploit the unique features of triangular meshes for direct 3D shape analysis using a convolutional neural network. Like traditional CNNs, MeshCNN involves creating convolution and pooling layers on the edges of 3D meshes, which enables the utilization of standard convolutional neural network tools. As a result, they were able to achieve an accuracy of 98.6% on 30 classes from the SHREC 11 dataset, although accuracy results for ModelNet40 were not reported. Additionally, the method shows remarkable segmentation performance on datasets related to human body and object parts.

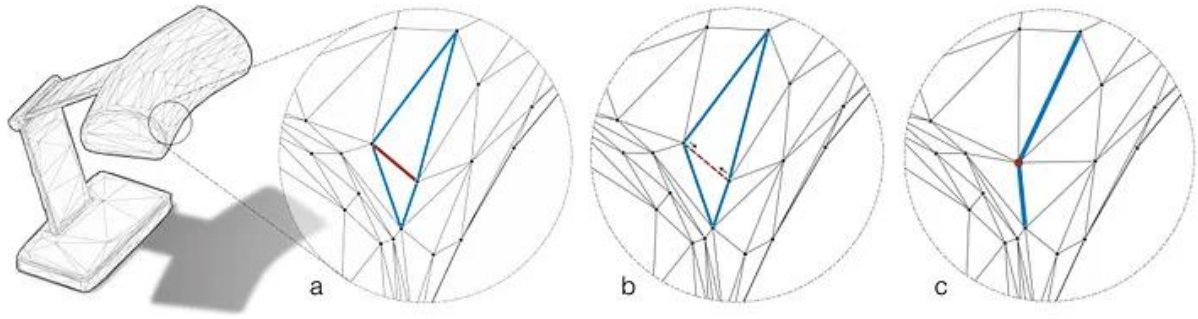

Figure 1: *The mesh pooling operation can be depicted in three steps. In the initial triangle mesh, a specific edge (colored in red) is shown to have four neighboring edges (colored in blue). The pooling operation begins by dissolving the selected edge, which involves merging its two vertices. This merging process, in turn, merges the neighboring edge pairs on either side of the dissolved edge. This results in a new mesh with two edges, as shown in the final image. This process is illustrated in a figure from the MeshCNN paper.*

The edges of a 3D mesh that is completely sealed are connected to two faces, except for non-manifold or boundary edges. If the mesh consists only of triangles, then two adjacent faces connected to an edge will have a total of five edges in common. This regularity makes triangular meshes highly suitable for machine learning applications. MeshCNN assumes that all models are manifold triangular meshes due to this regularity. Triangular meshes are often produced by popular 3D reconstruction methods such as photogrammetry, structured light scanning, and laser scanning, making this approach

applicable to such data. Authored meshes usually contain quads or n-gons, but they can be conveniently converted to triangular meshes using software programs like Maya or Blender.

By defining a convolution over an edge and its four neighbors, we can build a convolutional neural network that processes the entire mesh. However, the challenge is to define a set of operations that meet the following requirements:

- They must be invariant to vertex or edge order, whether locally or globally.
- They must be invariant to similarity transformations such as mesh translation, rotation, and scale.
- They must convey the relationship between a given vertex or edge and its neighbors, as well as its position in the global structure of the mesh.

a. Mesh Convolution

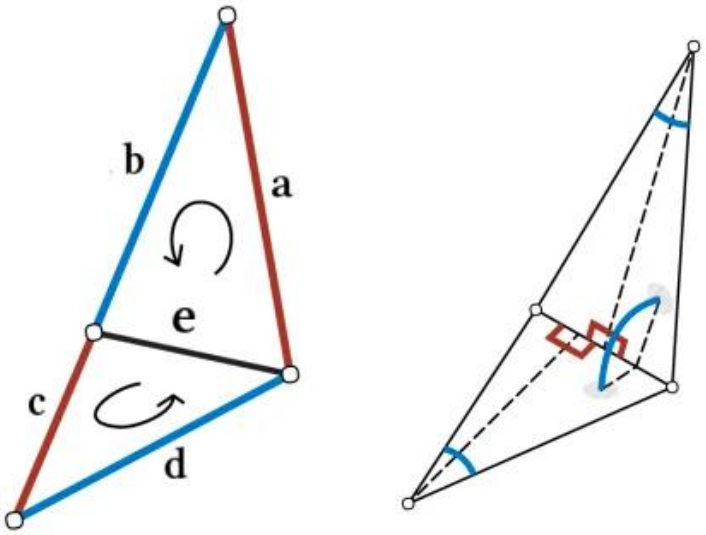

Figure 2: *The image in Figure 4 of the MeshCNN paper shows an edge and its neighbors on the left. The edges a, c (red) and b,d (blue) are opposing pairs, and their order is strictly counter-clockwise with respect to the center edge e. The input features for a given edge on the right are also designed to be invariant to the ordering of edges.*

The convolution operation in MeshCNN is designed to be invariant to the ordering of edges and their features. When a convolution is performed on an edge and its four neighbors, which all have their own features, the ordering of these edges must not affect the result. To achieve this, the authors use symmetric functions that process pairs of

opposing edges, such as a, c and b, d. These edge pairs are ordered in a counter-clockwise manner relative to the center edge e, and their finite sums and differences are computed before the convolution is applied as follows:

$$(e^1, e^2, e^3, e^4) = (|a - c|, a + c, |b - d|, b + d)$$

```
class MeshCov(nn.Module):
  def __init__(self, in_c, out_c, k=5, bias=True):
    super(MeshConv, self).__init__()
    self.conv = nn.Conv2d(in_channels=in_c, out_channels=out_c, kernel_size=(1, k), bias=bias)


  def forward(self, x)
    """
      Forward pass given a feature tensor x with shape (N, C, E, 5):
        N - batch
        C - # features
        E - # edges in mesh
        5 - edges in neighborhood (0 is central edge)
    """
    x_1 = x[:, :, :, 1] + x[:, :, :, 3]
    x_2 = x[:, :, :, 2] + x[:, :, :, 4]
    x_3 = torch.abs(x[:, :, :, 1] - x[:, :, :, 3])
    x_4 = torch.abs(x[:, :, :, 2] - x[:, :, :, 4])
    x = torch.stack([x[:, :, :, 0], x_1, x_2, x_3, x_4], dim=3)
    x = self.conv(x)
    return x
```

Figure 3: *Code for equivariant convolution.*

b. Mesh Input Features

One important issue that needs to be addressed is the selection of features to use as input for the convolution operation. Like the RGB channels in 2D images, an input feature representation must be created before applying the first convolution. To address this, the authors of the MeshCNN paper define five input features for each face, including the dihedral angle (the angle between two adjoining faces), the symmetric opposite angles (sorted to maintain order invariance), and two edge-length ratios (height/base ratio for each triangle, also sorted).

```
def dihedral_angle(mesh, edge_points):
```

```
    """
    Angle between two faces connected to edge.
    """
    normals_a = get_normals(mesh, edge_points, 0)
    normals_b = get_normals(mesh, edge_points, 3)
    dot = np.sum(normals_a * normals_b, axis=1).clip(-1, 1)
    angles = np.expand_dims(np.pi - np.arccos(dot), axis=0)
    return angles

def symmetric_opposite_angles(mesh, edge_points):
    """
    Angles of opposite corners across edge.
    """
    angles_a = get_opposite_angles(mesh, edge_points, 0)
    angles_b = get_opposite_angles(mesh, edge_points, 3)
    angles = np.concatenate((np.expand_dims(angles_a, 0), np.expand_dims(angles_b, 0)), axis=0)

    angles = np.sort(angles, axis=0)
    return angles

def symmetric_ratios(mesh, edge_points):
    """
    Height/base ratios of the triangles adjacent to edge.
    """
    ratios_a = get_ratios(mesh, edge_points, 0)
    ratios_b = get_ratios(mesh, edge_points, 3)
    ratios = np.concatenate((np.expand_dims(ratios_a, 0), np.expand_dims(ratios_b, 0)), axis=0)

    return np.sort(ratios, axis=0)
```

Figure 4: *Code for input features.*

c. Mesh Pooling

The pooling operation in MeshCNN involves merging the two vertices of an edge together, resulting in the 5 edges of a neighborhood being collapsed into 2. The features of the new edges are calculated as the average of the features of the original edges. For example, if we merge the two vertices of the edges a, b, c, d, and e, the new edges will have features equal to $avg(a, b, e)$ and $avg(c, d, e)$. The edges to be collapsed are

determined by sorting the edges based on the squared magnitude of their features and iteratively collapsing them until the desired number of edges is achieved.

To determine which edges to collapse during the pooling operation, a target number of edges is specified for each mesh pooling layer using the --pool_res argument in the code. The mesh pooling layer sorts the edges by the squared magnitude of the edge features and then iteratively collapses the mesh edges until the target number of edges is reached.

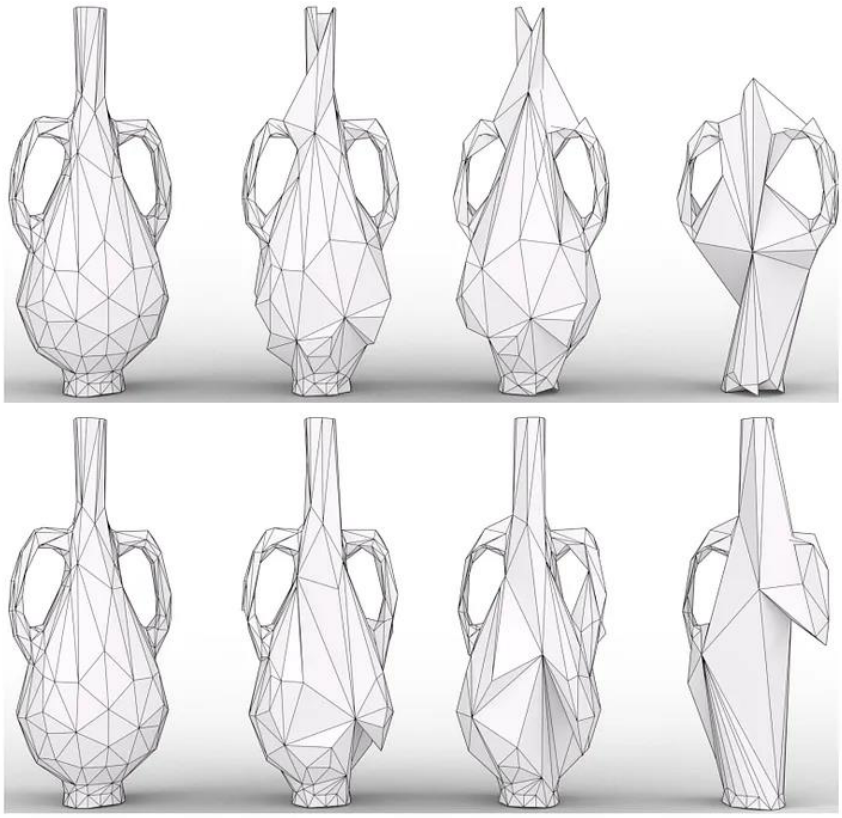

Figure 5: *According to Figure 1 of the MeshCNN paper, mesh pooling can preserve the edges that are most relevant to a particular task. This is demonstrated by the top image, which shows that pooling for the "handle or no handle" task preserves the handles in the vase, while the bottom image shows that pooling for the "neck or no neck" task retains the neck of the vase. In other words, mesh pooling learns to identify and keep the edges that are most informative for a given task.*

MeshCNN enables the model to learn weights that optimize for a specific task by using a learned mesh pooling operation. To revert the pooling operation, a mesh unpooling operation is required, which is crucial for the segmentation task. Therefore, the network

must keep track of the pooling operations performed in the encoder stage of the U-Net style segmentation network. MeshCNN accomplishes this by tracking the edge collapse operations in the history_data attribute of the mesh.

d. Architecture

The network architecture of MeshCNN is built using MResConv layers. Each layer comprises an initial mesh convolution (MeshConv), a series of ReLU+BatchNorm +MeshConv layers, and a residual connection that leads to another ReLU. The pattern of MResConv+Norm+MeshPool is repeated multiple times before the task layer is reached. For classification tasks, the task layer involves global average pooling and two fully connected layers. The segmentation network follows a U-Net style encoder-decoder structure.

3. 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:

i. 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,

ii. 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.

iii. You also need to come up with a new approach for mesh pooling that can be based on merging edges, vertices, or faces,

iv. 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.

v. 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.

vi. You need to explain how you overcame the problem of the vanishing gradient in your new CNN architecture.,

vii. To avoid overfitting, you need to implement techniques such as dropout, regularization, or early stopping,

viii. 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.

ix. 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.

x. *Innovations based on voxels, 2D images, and point clouds are not accepted for this challenge.*

4. Assessments

The assessment of this challenge will be based on several criteria including the relevance of the innovation proposed by the team, the quality of the solution provided, the obtained scores in terms of accuracy and other relevant metrics, as well as the thoroughness and clarity of the explanation provided for each task mentioned in Section 3. Other factors that may be considered include the creativity and originality of the approach, the efficiency and scalability of the model, and the overall quality of the report and code

submission. It is important for the team to clearly explain their thought process, the steps they took to address each task, and the results obtained. The judges will evaluate the submissions based on these criteria and determine the final scores and rankings accordingly.

The team that generates an exceptional innovation with the highest scores will receive assistance in publishing their work in a scientific journal and will be awarded a score greater than 90%.

5. Submission

To complete this challenge, you are required to submit a compressed .zip file containing all programmed solutions. Along with the solutions, you must also submit a separate report in PDF format. The report should describe the contributions of each team member and provide a detailed explanation and interpretation of the requirements listed in Section 3, as well as the obtained scores.

6. References

1. Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., & Cohen-Or, D. (2019). MeshCNN: a network with an edge. ACM Transactions on Graphics (TOG), 38(4), 1-12.
2. Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., & Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. IEEE Signal Processing Magazine, 34(4), 18-42.
3. Masci, J., Rodolà, E., Boscaini, D., Bronstein, M. M., & Li, H. (2016). Geometric deep learning. In SIGGRAPH ASIA 2016 Courses (pp. 1-50).
4. Video lecture: Deep Learning on Meshes – Rana Hanocka.

Good luck :)