NormalNet

Using PointNet for Normal Estimation in 3D Point Clouds

Ty Feng, Yasaman Hajnorouzali, Mahdis Rabbani

1 Problem and Motivation

We live in an inherently three-dimensional world. With the increasing availability of sensors and data acquisition tools such as LiDAR as well as commercially available photogrammetry software, it is increasingly feasible to capture 3D data of scenes and objects around us. Capturing the world in 3D has numerous applications, including creating AR games and applications, 3D mapping, and autonomous driving. Among all types of 3D data, point clouds are the most basic and the closest to raw sensor data. As a canonical form of 3D data, point clouds can be further processed into 3D meshes, a process known as surface reconstruction or meshing [1, 5]. Creating 3D meshes that accurately captures the geometric structures of objects is highly dependent on the quality of the point clouds and its point normals. In many cases, point normals were not collected by sensors during 3D data acquisition, and they have to be estimated based on local geometry. Previously, normal estimation for point clouds was done by fitting neighboring points with a plane using principal component analysis (PCA) and kNN [4]. However, the requirement of manually selecting a neighborhood size is challenging in practice – if the neighborhood size is too small, the normal estimates may be unstable in the presence of noise; if the neighborhood size is too big, it may over-smooth the reconstruction and leave out finer details. In the PointNet paper [2], the authors have shown for the first time they could consume point clouds directly into their deep neural network for object classification and segmentation without the need of voxelization nor rendering 3D points to 2D images. We would like to use PointNet for normal estimation of 3D point clouds by modifying PointNet's segmentation network to learn point normals instead of object categories.

2 Prior Work

2.1 Deep Learning on Point Clouds

The point cloud is a canonical 3D data format, as it can be converted to other formats (eg. meshes) easily. Most researchers convert this type of data to standard 3D voxel grids or 2D image projections for deep learning due to its unordered and irregular format. However, voxelization makes data needlessly voluminous and 2D projection can lose the 3D geometric information. In 2017, Qi et al. in their PointNet paper have shown for the first time that 3D point clouds can be used directly as the input to a deep neural network. One significant contribution of the PointNet paper is their solution to the problem that the neural network should be invariant to the input order of 3D points. 3D point clouds cannot be simply fed into the network as a list of x,y,z points since doing so would create models trained on that specific input order. PointNet used maxpooling as the symmetric function to learn a canonical global feature vector that allows the network to be invariant to point input order.

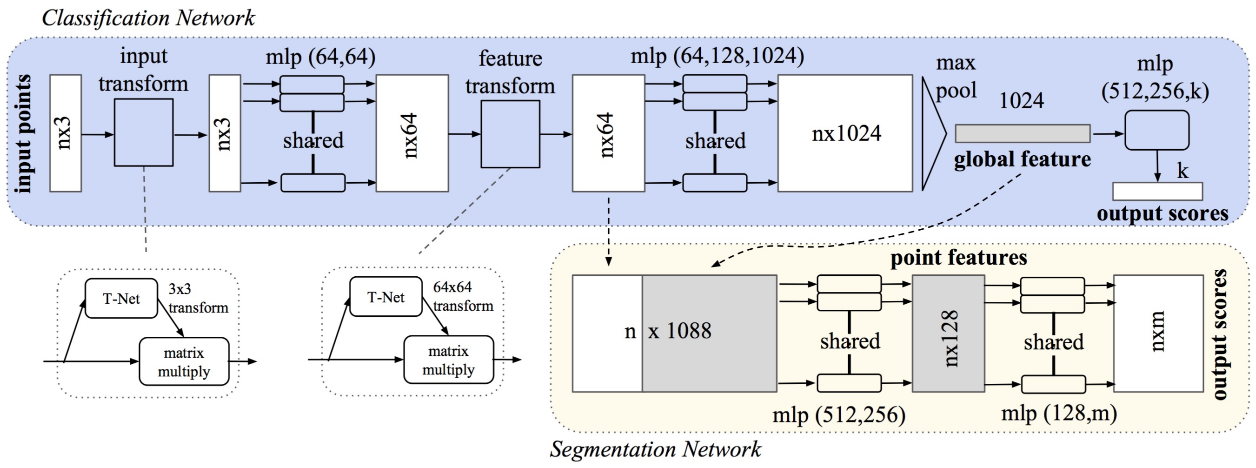


Figure 1: The architecture of PointNet. The classification network receives n points as input, transforms the input and features, and then pools all of the points features together. The results are m classification scores. The classification net is expanded upon by the segmentation network. It combines local and global elements, as well as outcomes measured in points. Multi-layer perceptron is abbreviated as mlp, and the numbers in brackets represent its layer sizes. All layers using ReLU employ the batch norm method. In a classification net, dropout layers are employed for the final mlp.

2.2 PointNet

Utilizing a single symmetric function, max pooling, to handle input sets with unordered elements is the key contribution of PointNet. In practice, the network picks out interesting or instructive points from the point cloud and encodes the rationale behind its selection using a set of optimization functions/criteria. The network's final fully connected layers combine these learned optimum values into a global descriptor for the entire point cloud for a shape classification or to predict per-point labels (shape segmentation). To further enhance the results, PointNet applies an input spatial transform that makes an effort to canonicalize the input before it is fed into the rest of the network.

Three methods can be used to make a model invariant to input permutation:

- 1) Sort input using canonical order
- 2) To train an RNN, treat the input as a series, but add additional training data with various permutations.
- 3) Utilize a straightforward symmetric function to combine the data from each point.

A symmetric function is used in this case to take vectors as input and produce a new vector that is independent of the input order. The operators + and * are two instances of this function.

2.3 Normal Estimation

Normal estimation in 3D point clouds means that for each point, we calculate its normal that best describes the local surface. Accurate point normals are crucial for many surface reconstruction algorithms that rely on point normals to reconstruct a mesh. Principal Component Analysis (PCA) is one of the most straightforward and widely used [4]. The problem of estimating the normal of a plane tangent to the surface, which then transforms into a least-square plane fitting estimate problem, approximates the normal to a point on the surface. Normal estimation using PCA is sensitive to challenges from the real-world data, such as outliers and noise, and does not maintain sharp features. Additionally, it is difficult to hand pick the point neighborhood size for PCA plane-fitting.

3 Data

We used the PCPNet dataset [3] of 3D scans of figurines and man-made objects. Each shape has 100000 points. Ground truth normals were derived from the normal vector for each triangle in the shape's original triangle mesh. We also used their data loader and samplers in our project.

4 Method

4.1 Our NormalNet Architecture

We followed the PointNet architecture for the computation of global point set features and local point features. The input is a nx3 point cloud tensor with x, y, z points. For each point, we use PointNet to compute its local point feature by feeding the nx3 point set through multi-layer perceptron layers, which outputs a nx1024 feature tensor. Then the symmetric operation is applied to all of the point feature tensors to create a global feature vector (1024). Then we concatenate the local point features (nx1024) with the global feature vector to be used as the input (nx2048) for our NormalNet. The intuition behind this concatenation of local and global features is that point normals are local geometric properties that have to be informed by the local point features. Using only the global feature of a shape may not work so well for predicting local geometric properties. Then we used six fully-connected linear layers with batch normalization, and ReLU to output predicted normals (nx3). Each layer reduces the feature dimension from the previous layer by half. In the middle layers, we used dropout to reduce the chance of overfitting. We used cosine similarity, $cosSimilarity(v1, v2) = \frac{v1 * v2}{||v1||||v2||}$, in the loss function, $1 - abs(cosSimilarity(v1, v2))$. It has good interpretability for comparing the angle distance between predicted and ground truth normal vectors.

We also changed the symmetric function in PointNet's global feature computation to sum, instead of maxpooling. The results are discussed in the experiments section.

5 Experiments and Results

We trained for 100 epochs for the following experiments on a Nvidia V100 GPU on Google Cloud, which took 1.5 hours for each training run. We used 0.0001 as the learning rate with 0.9 as the momentum. We also used L2 regularization by setting weight decay in SGD to 0.01 to prevent overfitting. We compared using maxpooling or sum as the symmetric function in PointNet's global feature computation. The original PointNet paper used maxpooling as their symmetric operation. We wanted to see if changing this symmetric operation to sum would have any effect. Using sum gave slightly better test loss at epoch 100, compared with using maxpooling. The losses are plotted

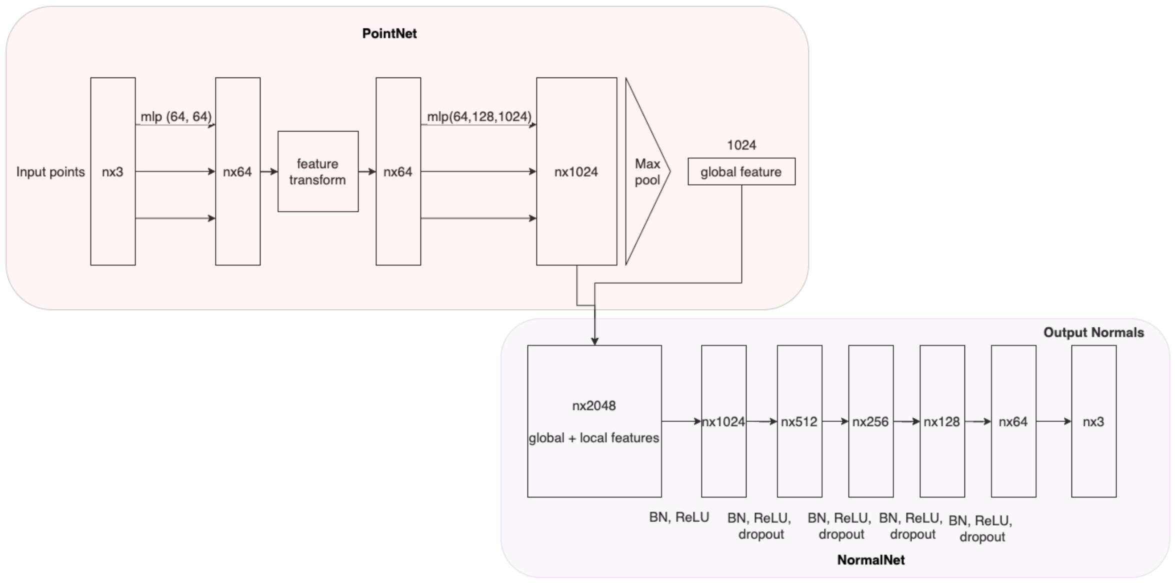


Figure 2: Our NormalNet architecture. We used the PointNet architecture for computing both the global and local point features, which are concatenated to be used as input in NormalNet. Our NormalNet has six fully-connected linear layers with batch normalization and ReLU. We also used dropout layers to reduce the chance of overfitting in the latter layers. The output is a nx3 normal vector.

in Figure 3.

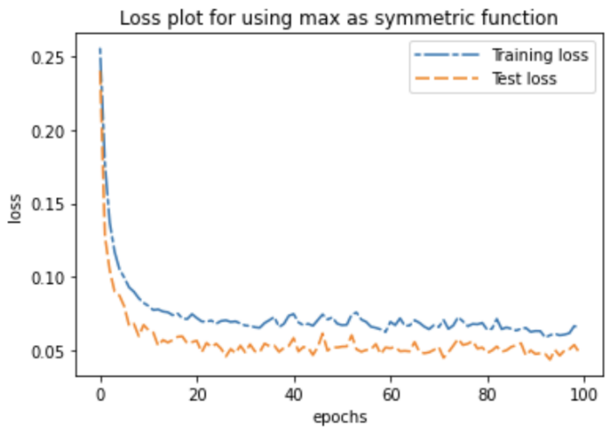
We also compared between using global features from PointNet only and using both local per point features and global features. The results validate our choice of concatenating the local and global features before inputting it into our network. At epoch 100, using global features only results in a test loss of 0.063. However, using both local and global features as well as sum as the symmetric function results in a test loss of 0.043. These losses are plotted in Figure 4.

6 Conclusion

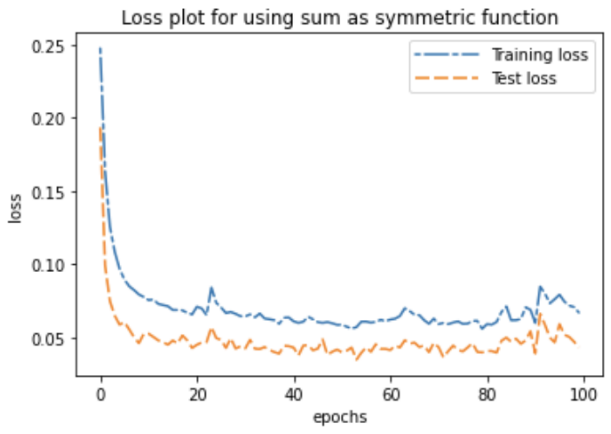
In conclusion, our main contributions are the following:

- We developed a six-layer network, NormalNet, that outputs point normals
- We concatenated local (per point) and global features from PointNet and used it as our input
- We used sum as the symmetric function in PointNet's global feature computation

Our results support our design choices.



```
NormalNet using max as symmetric function test loss at epoch 100: 0.04837148778140545
```



```
NormalNet using sum as symmetric function test loss at epoch 100: 0.043209195137023926
```

Figure 3: Loss plots for comparing max and sum symmetric functions

7 Contributions

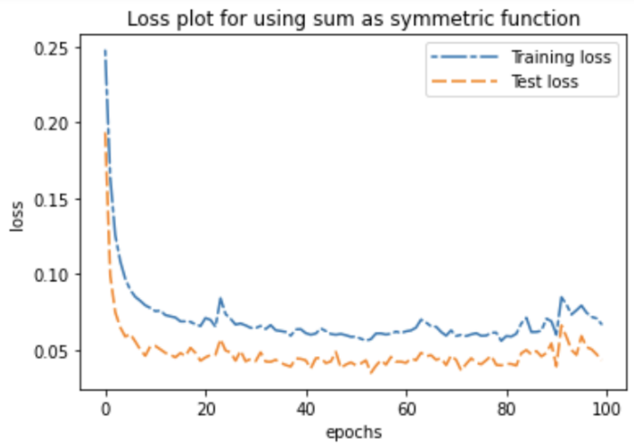
Ty Feng wrote the code and contributed to the report. The literature review was completed by Mahdis Rabbani and Yasaman Hajnorouzali, who also made contributions to the part on past work, worked to debug the code, and helped with the creation of the presentation slides.

References

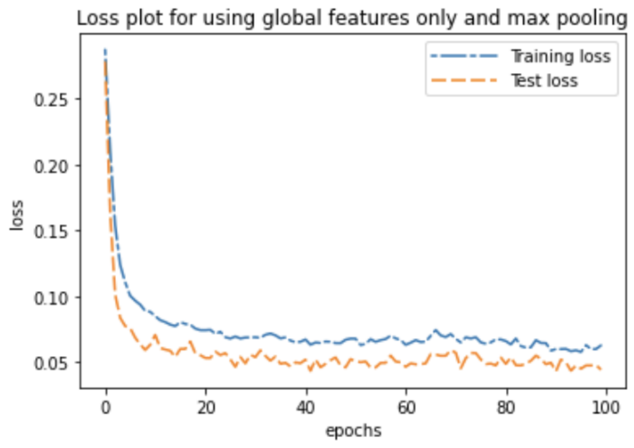
[1] Fausto Bernardini, Joshua Mittleman, Holly Rushmeier, Claudio Silva, and Gabriel Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, Oct 1999.

[2] Qi Charles, Hao Su, Mo Kaichun, and Leonidas J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[3] Paul Guerrero, Yanir Kleiman, Maks Ovsjanikov, and Niloy J. Mitra. PCPNet: Learning local



```
NormalNet using sum as symmetric function test loss at epoch 100: 0.043209195137023926
```



```
NormalNet using global feature only's test loss at epoch 100: 0.06289172423258424
```

Figure 4: Loss plots for comparing using global features only vs using both global and local features

shape properties from raw point clouds. *Computer Graphics Forum*, 37(2):75–85, 2018.

[4] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH Comput. Graph.*, 26(2):71–78, July 1992.

[5] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, page 61–70, Goslar, DEU, 2006. Eurographics Association.