CS 282A Deep Neural Networks

Homework : Implementation of the PointNet Architecture

Homework Designer: Arm Wonghirundacha, Joohwan Seo, Seunghoon Paik

1 Understanding PointNet Architecture

We introduce PointNet[2], a novel neural network architecture that directly processes point cloud data for tasks such as image classification, image segmentation and semantic segmentation, where we see examples of these tasks in Figure 1.

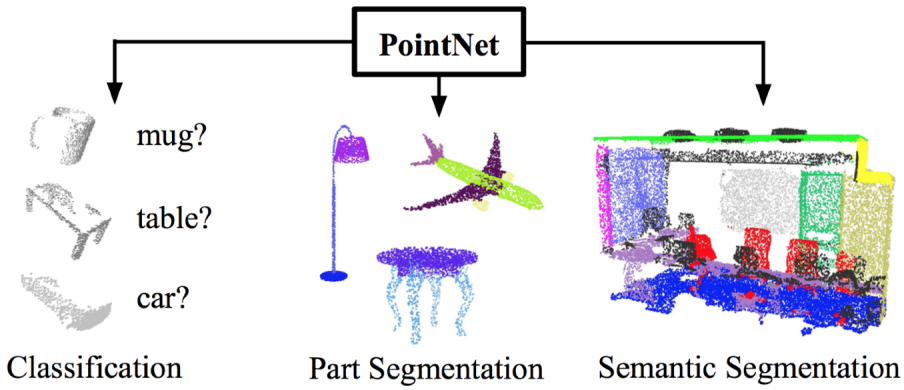


Figure 1: Tasks for PointNet

In order to understand the motivation behind the architecture, we need to understand why point clouds are a desirable three-dimensional representation. Firstly, three-dimensional sensors such as LiDAR (Light Detection and Ranging) and Depth sensors record data that are in close similarity to point cloud data. They are also a simple and unified structure that avoids the combinatorial irregularities and complexities of meshes, making them easier to learn from. Point clouds also can represent a wide range of geometric structures, including irregular shapes and surfaces, which may be difficult to capture with other data types like voxel grids or images. Lastly, point clouds are inherently unordered sets of points, which makes them permutation invariant and thus more flexible for processing with neural networks.

With this in mind, we can see why learning from point clouds is beneficial. The primary challenge to learning from point clouds is that the model needs to be invariant to permutations. This is due to the unordered nature of the data. We also have to ensure the model has invariance under geometric transformations, meaning point cloud rotations should not alter classification results.

These two main challenges lead us to the architecture design choices in PointNet; the use of symmetric functions (max pooling) for permutation invariance, and the learning of a spatial transformer network (T-net) for invariance under geometric transformations. Both of these choices can be seen in Figure 2.

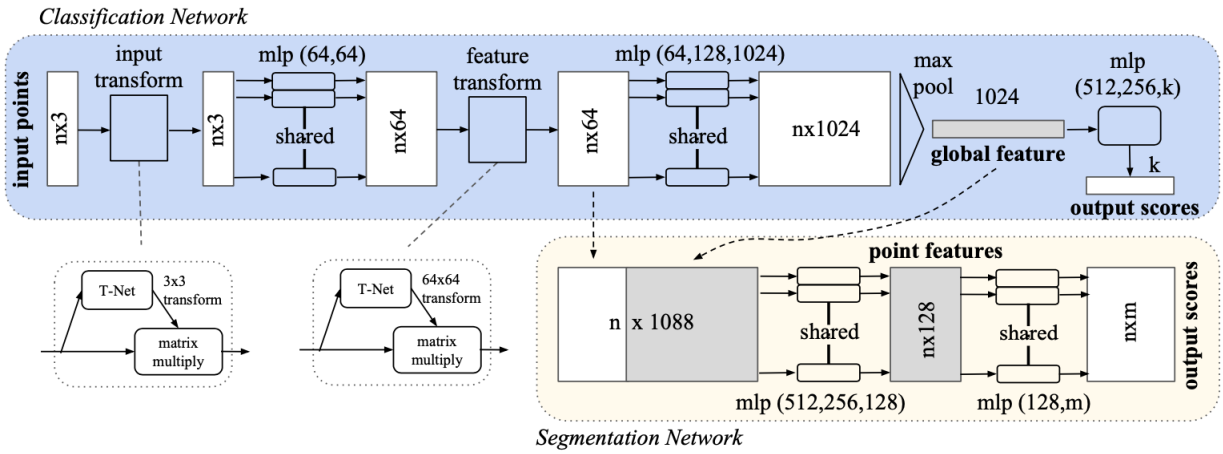


Figure 2: PointNet Architecture

1.1 Permutation Invariance

A critical concept of the Pointnet is permutation invariance: the results of the Pointnet should not change as the sequence of the points changes. Consider, for example, a set of 3 points, $p_i = [x_i, y_i, z_i]^T$, for $i = 1, 2, 3$. Then, the outputs of the Pointnet function $f(\cdot)$ should satisfy:

$$f(p_1, p_2, p_3) = f(p_1, p_3, p_2) = \cdots = f(p_3, p_2, p_1)$$

Question 1.
Select all permutation invariant functions and note the reason why briefly. You may present a counterexample if the function is not permutation invariant. Note that without loss of generality, $x_i \in \mathbb{R}^n$, $f_j : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}^n$

1. $f_1(x_1, x_2) = \frac{1}{2}(x_1 + x_2)$
2. $f_2(x_1, x_2) = \max(x_1, x_2)$
3. $f_3(x_1, x_2) = w_1 x_1 + w_2 x_2$, $w_1, w_2 \in \mathbb{R}$, $w_1 \neq w_2$
4. $f_4(x_1, x_2) = (a^T x_1) x_2$, $a \in \mathbb{R}^n$
5. $f_5(x_1, x_2) = w_1 x_1 + w_1 x_2$, $w_1 \in \mathbb{R}$

(Sol) 3 and 4 are not permutation invariant. For 3, since the weight is not shared, the output is not the same, and for 4, a counter-example is where x_1 is orthogonal to a, but x_2 is not orthogonal to a.

1.1.1 Weight sharing for permutation invariance

As you can see in the previous part, weight-sharing is needed to guarantee the permutation invariance of the PointNet. There is an easy way to implement the weight-shared linear layer in PyTorch- `torch.nn.Conv1d`, and we will implement the code using this function.

Question 2.
Please refer to `https://pytorch.org/docs/stable/generated/torch.nn.Conv1d.html` for the usage of

`Conv1d` and the notations. Suppose that our $C_{in} = 3$, the Euclidean coordinates, $C_{out} = 64$, $L = 1$, $N = n_{batch}$.

1. Briefly describe how `Conv1d` work as the weight-shared fully connected linear layer.

   (Sol) `Conv1d` with the kernel size 1 and stride value 1 aggregates the channel information and outputs the a scalar. Since there are 64 kernels, the output size will be $N \times 64$, where the input size is $N \times 3$. Therefore, if we make the size comparison, we can also understand that `Conv1d` is mapping 3 to 64 for each batch - which works as a linear layer. In this sense, `Conv1d` provides a single-line implementation for the weight-shared linear layer.

2. For `Conv1d` to work as the weight-shared fully connected linear layer, what should be the value of stride and kernel size?

   (Sol) Filter size and the stride should be 1.

1.2 Invariance under geometric transformations

The T-net is neural network architecture used in PointNet to perform the alignment and transformation of input points and features. T-Net consists of a small multi-layer perceptron (MLP) followed by a matrix multiplication layer that outputs a 3×3 or 64×64 transformation matrix. This matrix is then applied to the input points or features to perform alignment and transformation. The T-Net is used twice in PointNet: once for input point alignment/transformation (T1) and once for feature alignment/transformation (T2).

Figure 3 shows us the effect of a learned affine transformation matrix on an input image.

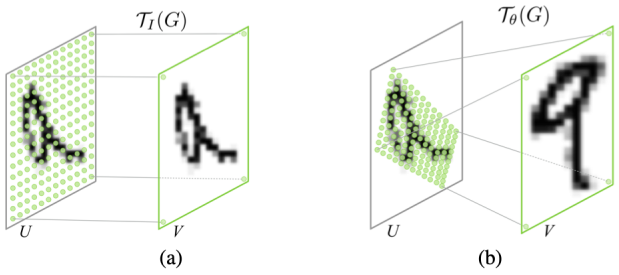


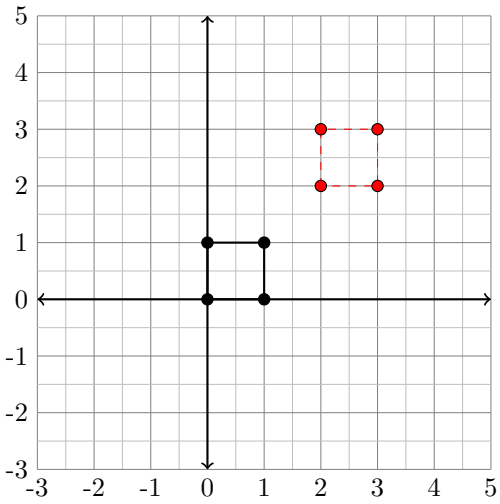
Figure 3: Applying an affine transformation [1]

In order to better understand what the T-net is learning, we will work through a short example of transformation matrices in two dimensions.

We take the transformation matrix multiplied on the left to the vertex of the matrix to get the resulting transformed vertex point. An example of this is given below. The first two elements of the vertex matrix represent the (x, y) coordinate, while the third element represents the transformation value. Below we see examples of translation and shearing of the unit square using a transformation matrix.

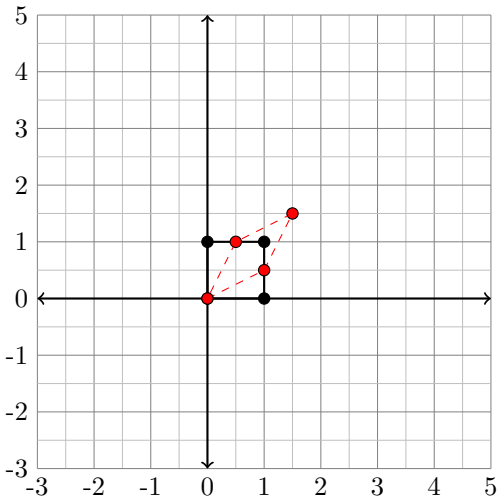
These examples only contain simple affine transformations, however spatial transformers networks like T-Net can learn many different transformations such as plane projective transformations, piecewise affine, or thin plate spline [1].

Matrix Translation



- $\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 2 \\ 1 \end{pmatrix}$

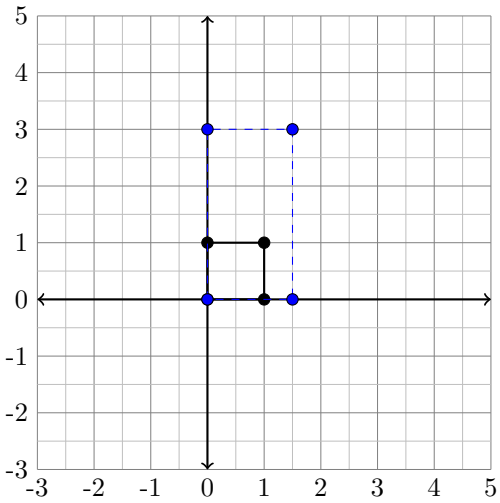
Matrix Shearing



- $\begin{pmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 1.5 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0.5 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0.5 & 0 \\ 0.5 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 1 \\ 1 \end{pmatrix}$

Question 3a. Calculate the transformed vertices and plot them on the graph.

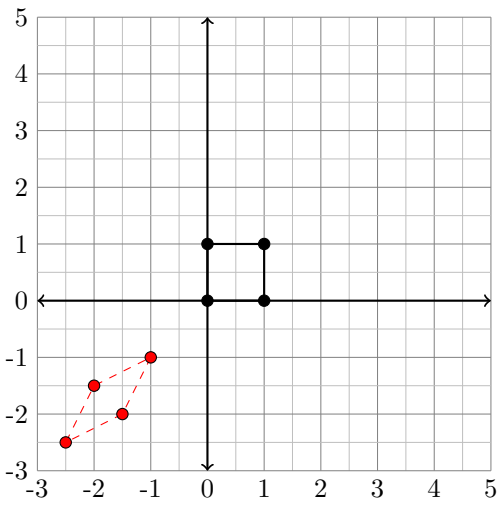
Matrix Scaling



- $\begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 3 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.5 \\ 0 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ 1 \end{pmatrix}$

Question 3b. Calculate the transformation matrix to get the resulting vertices.

Matrix shear and translate



- $\begin{pmatrix} 1 & 0.5 & -2.5 \\ 0.5 & 1 & -2.5 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -2.5 \\ -2.5 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0.5 & -2.5 \\ 0.5 & 1 & -2.5 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0.5 & -2.5 \\ 0.5 & 1 & -2.5 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} -1.5 \\ -2 \\ 1 \end{pmatrix}$
- $\begin{pmatrix} 1 & 0.5 & -2.5 \\ 0.5 & 1 & -2.5 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ -1.5 \\ 1 \end{pmatrix}$

2 Implementation of the PointNet Code

For the coding part, we will implement the classification and part segmentation models for the PointNet Architecture.

Question 4. In the final part of the jupyter notebook file, attach the training loss and accuracy plot in the written part of the homework, together with the test accuracy.

References

[1] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks, 2016.

[2] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.