Tampere University

Hoanh Le

GEOMETRIC INVARIANCE OF POINTNET

Bachelor of Science Thesis
Faculty of Information Technology and Communication Sciences
Supervisor: Prof. Joni Kämäräinen
June 2021

ABSTRACT

Hoanh Le: Geometric Invariance of Pointnet
Bachelor of Science Thesis
Tampere University
Science and Engineering
June 2021

PointNet has become one of the de facto deep learning architectures for 3D tasks, from object classification to scene segmentation. One of its main components is two Joint Alignment Networks, which were designed to help PointNet to be invariant to geometric transformation such as rigid transformation. They attempt to canonicalize the input set and feature space before feeding them to the main network. However, their effects have not been studied extensively. In this work, we will evaluate PointNet's performance in the presence or absence of Joint Alignment Networks under rotation transformation. We show that with a limited number of data, the use of Joint Alignment Networks does not increase Pointnet's robustness against rotation transformation but can actually decrease it.

Keywords: PointNet, 3D, Spatial Transformation Network, Joint Alignment Network, geometric invariance, rotation invariance

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

Over recent years, deep learning has allowed the computer vision community to have many remarkable advances and achieve state-of-the-art results on several 2D tasks: classification [1, 2, 3], object detection [4, 5], semantic segmentation [6, 7], and scene understanding [8], amongst others. Driven by that success, the application of deep learning in 3D tasks became one of primary targets in current computer vision research.

Unlike 2D data that has only one representation, a matrix of pixels, 3D data involves a multitude of representations which lead to various approaches. PointNet [9] has been the pioneering and backbone architecture that processes raw point cloud, and it has attracted much attention from the research community by its promising results. One of the main components in PointNet architecture is the Joint Alignment Network, which attempts to canonicalize the data before the feature extractor processes it. However, its performance against geometric transformation has not yet been studied in detail.

In this work, we will analyse the performance of PointNet with and without Joint Alignment Network under geometric transformation in the object classification task. By using data normalization, PointNet can be invariant to the effect of translating and scaling the point set. Therefore, we chose to only test its performance under rotation. By using different rotation ranges in different axes (x, y, z, and arbitrary), we saw that PointNet with Joint Alignment Network is only invariant to a small degree of rotation. The model's performance will decrease rapidly if the rotation range widens.

The remainder of this thesis is arranged as follows. We started with related work in Chapter 2. Chapter 3 explains how we test PointNet's geometric invariance ability and the experiment's results. Finally, chapter 4 presents the summary of this thesis and discussion regarding possible future work.

2. RELATED WORK

2.1 PointNet

Since PointNet consumes raw point cloud directly, its architecture needs to satisfy the following unique properties of point sets:

- Permutation invariance: since point sets are unstructured, the network that consumes *N* point sets must have the same results with *N!* permutations of the input point sets.
- Transformation invariance: the network output should remain constant if geometric transformation such as rotation or translation is applied to the input point sets.
- Neighboring points interaction: The network should be able to utilize useful information derived from interconnection between local points in space.

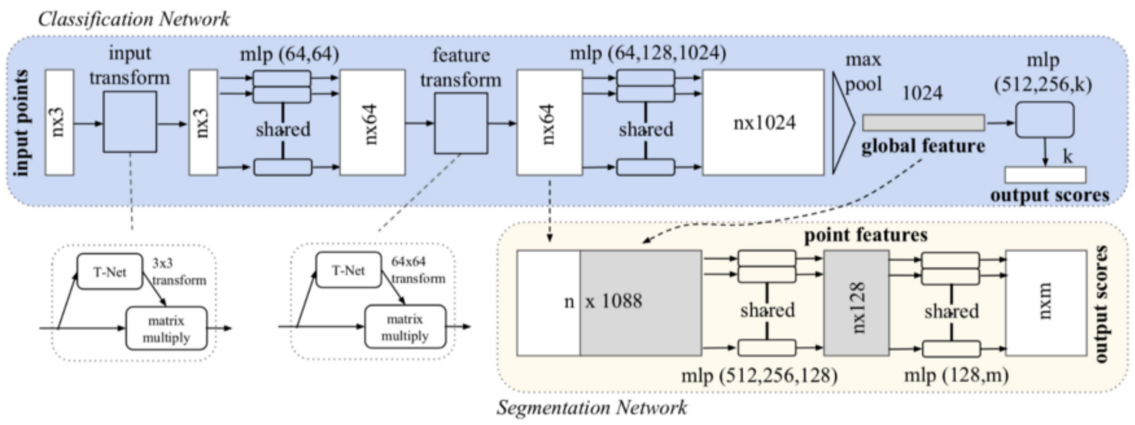


Figure 2.1. *PointNet architecture [9]*

PointNet is made up of three main components: a max pooling layer, a local and global information combination structure, and a Joint Alignment Network. Each module is designed to tackle each unique property of point cloud. However, the local and global information combination structure module is only used in the segmentation task, thus, it will be excluded from our analysis.

The following subsections will discuss the reasons for the max pooling layer and the joint alignment network and how they are used.

2.1.1 Max pooling layer

For the network to be invariant to permutation, Qi et al[9] use symmetric function: a function that takes n variables and outputs the same value regardless of the order of the input points.[10]

Common examples of symmetric function:

- $sum(x_1, x_2) = sum(x_2, x_1)$
- $max(x_1, x_2) = max(x_2, x_1)$
- $average(x_1, x_2) = average(x_2, x_1)$

Specifically, in PointNet architecture, it first maps the input data to a high-dimensional space by a multi-layer perceptron network [11], and then use a max pooling layer (symmetric function) to aggregate the global point features, as described in the blue part in Figure 2.1.

2.1.2 Joint Alignment Network

The classification of an input point set should be invariant to geometric transformation, such as rigid transformation. Motivated by Spatial Transformation Network [12], the Joint Alignment Network in PointNet (the input and feature transformation blocks in Figure 2.1) were designed to transform the data to a space expected by the multi-layer perceptron networks.

First, we take a closer look at Spatial Transformation Network to understand the adoption of this module in PointNet.

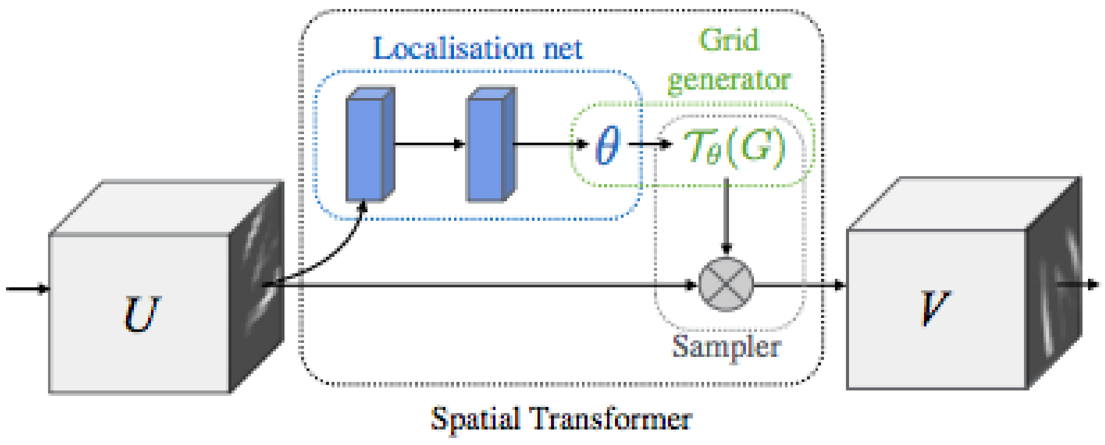


Figure 2.2. *Spatial Transformation Network[12]*

The network comprises 3 modules:

- A localization network, which takes an input feature map *U*, with height *H*, and width *W*, and output the transformation matrix T_0 to be applied to the original feature map *U*. The shape of the transformation matrix T_0 depends on the transformation type that we want the network to learn. For example, it can be constrained to only allow cropping, translation, and scaling as below

$$\mathrm{A}_\theta = \begin{bmatrix} s & 0 & t_x \\ 0 & s & t_y \end{bmatrix} \tag{2.1}$$

- A grid generator, which is used to calculate the target coordinates x_i^t, y_i^t, after we apply the transformation T_0 to the source coordinates x_i^s, y_i^s. Below is an example with a 2D affine transformation matrix.

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta\left(G_i\right) = \mathrm{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} \tag{2.2}$$

- A sampler, which takes the target set of coordinates x_i^t, y_i^t, along with the input feature map *U*, to produce the output feature map *V*

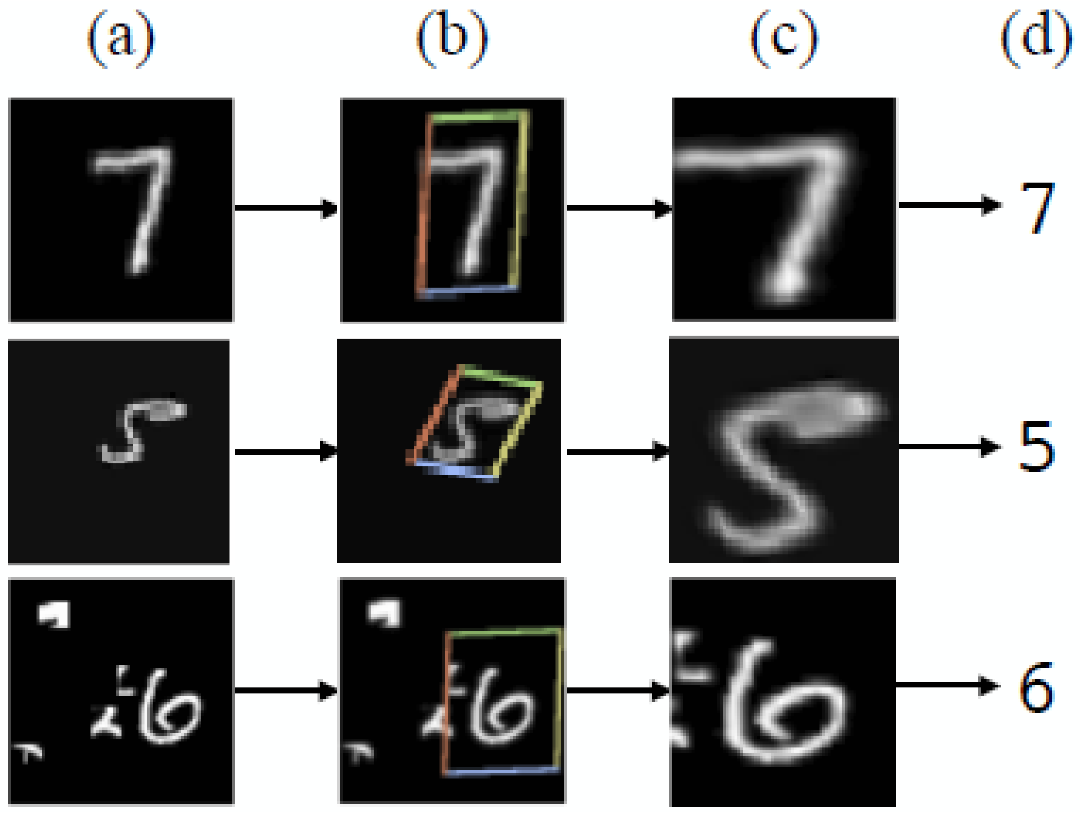


Figure 2.3. *Result of adding transformation network block in a fully-connected network in distorted MNIST dataset, with a is the input of the transformation network, b is the predicted transformation of the localization network, c is the output of the spatial transformation network, d is the prediction of the subsequent fully-connected network*

[12]

PointNet utilizes a similar approach in a much simpler way. Since the input data is point cloud, we can predict the transformation matrix by T-Net, which plays a similar role as the localization network in Spatial Transformation Network, and apply the transformation matrix directly to the point sets by matrix multiplication.

T-Net architecture (Table 2.1) bears a resemblance to the big network, and composes of different basic blocks: feature extractor, max pooling, and fully connected layer

Layer (type)	Output Shape	Param #
Conv1d-1	[-1, 64, 1024]	256
BatchNorm1d-2	[-1, 64, 1024]	128
Conv1d-3	[-1, 128, 1024]	8,320
BatchNorm1d-4	[-1, 128, 1024]	256
Conv1d-5	[-1, 1024, 1024]	132,096
BatchNorm1d-6	[-1, 1024, 1024]	2,048
Linear-7	[-1, 512]	524,800
BatchNorm1d-8	[-1, 512]	1,024
Linear-9	[-1, 256]	131,328
BatchNorm1d-10	[-1, 256]	512
Linear-11	[-1, 9]	2,313

Total params: 803,081

Trainable params: 803,081

Non-trainable params: 0

Table 2.1. *Model summary of T-Net with k = 3 (used to align input point cloud), and number of points N = 1024*

T-Net can also be extended to align the feature space described in the feature transform block in Figure 2.1, which will output a 64-by-64 transformation matrix. Due to the increase in the number of trainable parameters and difficulty in optimization, a regulation term is added to training loss. The transformation matrix is constrained to the proximity of the orthogonal matrix:

$$L_{reg} = \left\| I - AA^T \right\|_F^2 \tag{2.3}$$

2.2 PointNet++

One major drawback in PointNet is that it fails to capture information from the local neighboring points. To address this issue, Qi et al [13] introduced PointNet++, a hierarchical neural network based on the concept of CNN, in which features are captured on a bigger scale progressively through a multi-resolution hierarchy. PointNet++ first extracts local features from the small neighborhoods, and these local features are then aggregated into larger groups and processed to have higher-level features. This technique is repeated until the features of the whole point set have been obtained.

The hierarchical architecture is composed of 3 main components: Sampling layer, Group-

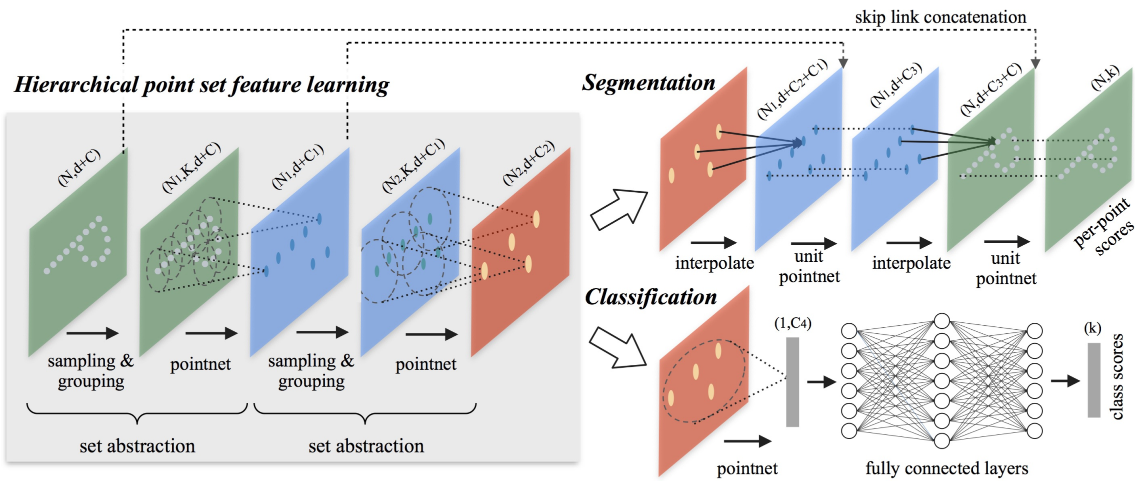


Figure 2.4. *Single-scale PointNet++ architecture[13]*

ing layer, and PointNet layer (Figure 2.4). First, the Sampling layer chooses a group of points from the input points to define the local region centroids. After that, the Grouping layer creates local region sets by locating neighboring points around the centroids. Finally, the PointNet layer adopts a mini-PointNet to encode local region structures into feature vectors. One thing to notice is that Qi et al [13] do not use the Joint Alignment Network module (Subsection 2.1.2) to align the input set to a canonical space before feeding it to the PointNet layer in PointNet++.

One more significant contribution in PointNet++ is density adaptive PointNet layers. With 2 different grouping methods (multi-scale grouping, multi-resolution grouping), PointNet layers are able to capture local patterns at multiple scales instead of a single scale (as in Figure 2.4), and combine them based on the point density in the local area.

- Multi-scale grouping (MSG): It applies the Grouping layers at multiple scales and then uses PointNets to extract features from each scale. A multi-scale feature is then formed by concatenating features at multiple scales.
- Multi-resolution grouping (MRG): Because the MSG technique performs local PointNet at large scale neighborhoods for every centroid point, it is computationally intensive. In the MRG technique, features of a region are concatenated of 2 vectors (Figure 2.5 b). One vector is produced by utilizing the selected abstraction to summarize the characteristics at each subregion from the lower level. The other vector is acquired by processing all raw points in the local region. By avoiding feature extraction in large scale neighborhoods, this method is more computationally efficient.

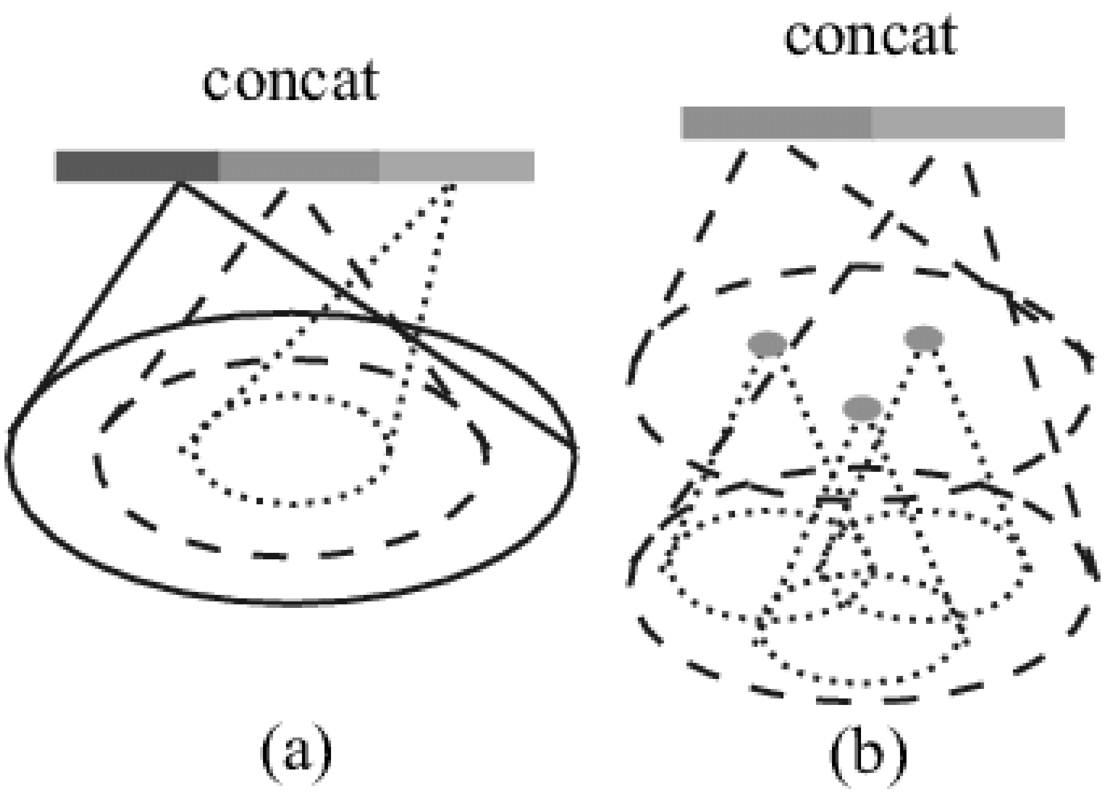


Figure 2.5. *(a) Multi-scale grouping (MSG); (b) Multi-resolution grouping (MRG)[13]*

3. EXPERIMENTS

The original implementation of PointNet was done using *Tensorflow* [14] library, but we chose to use *Pytorch* [15] library in this thesis. Pytorch is a fast growing, flexible, and easy to use machine learning framework with two high level features: tensor computation with strong GPU acceleration and deep neural networks built on an autograd system.

The method we use to test PointNet's geometric invariance ability is described in Section 3.1. The experiments were done using the ModelNet40 dataset [16] described in Section 3.2. The experiment results are described in Section 3.3. Finally, we discuss data augmentation in Section 3.4.

3.1 Method

The approach to test PointNet's geometric invariance in this thesis is described in 2 main conceptual components:

- Rotation in 3D
- Testing algorithm

3.1.1 Rotation in 3D

Rotation in 3D is more complicated than 2D. The axis of rotation in 2D is always perpendicular to *xy* plane, while in 3D, the axis of rotation can be arbitrary. We will look at 4 different matrix representations of main axes of rotation: x, y, z, and arbitrary.

A rotation matrix by angle θ about x, y, z axes can be described as follow:

$$
\begin{aligned}
R_x(\theta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \\
R_y(\theta) &= \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \\
R_z(\theta) &= \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{3.1}
$$

Euler's rotation theorem [17] states that any rotation in 3D can be represented as combination of rotation around 3 main axes: x, y, z. Therefore, a rotation matrix in arbitrary axis can be obtained by matrix multiplication of 3 rotation matrices. For example:

$$R = R_z(\alpha)R_y(\beta)R_x(\gamma) \tag{3.2}$$

$$R = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{3.3}$$

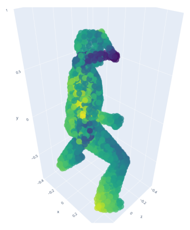
represents a rotation whose angles about x, y, z axes are α, β, γ respectively.

3.1.2 Testing algorithm

As previously mentioned, PointNet is invariant to scale and translation geometric transformation by normalizing the input point cloud. In order to test its rigid geometric invariance, we use the rotation transformation. The pseudocode for the algorithm is described in Listing 3.1

```
Given rotation_range [a, b], model F, and axis a

number_of_correct_samples = 0

for i in range(number_of_test_objects):
    test_object, label = take_next_object()
    original_predicted_label = model(test_object)

    if axis == arbitrary:
        rotation_angle = random.uniform(range = rotation_range,
   size = 3)
    else:
        rotation_angle = random.uniform(range = rotation_range,
   size = 1)
    rotated_object = rotate(test_object, angle = rotation_angle)
    rotated_predicted_label = model(rotated_object)


    if rotated_predicted_label == label:
      number_of_correct_samples += 1

rotated_accuracy = number_of_correct_samples /
   number_of_test_objects
```

Listing 3.1. *Algorithm to calculate the relative accuracy if we rotate the point cloud*


(a) *Original person*

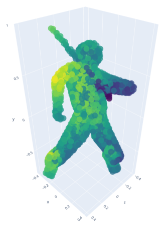

(b) *Rotated person*

Figure 3.1. *Illustration of rotating a 3D point cloud in y axis*

If PointNet is invariant to rotation transformation, we should expect that the accuracy will not drop too much going from small rotation range to wide rotation range.

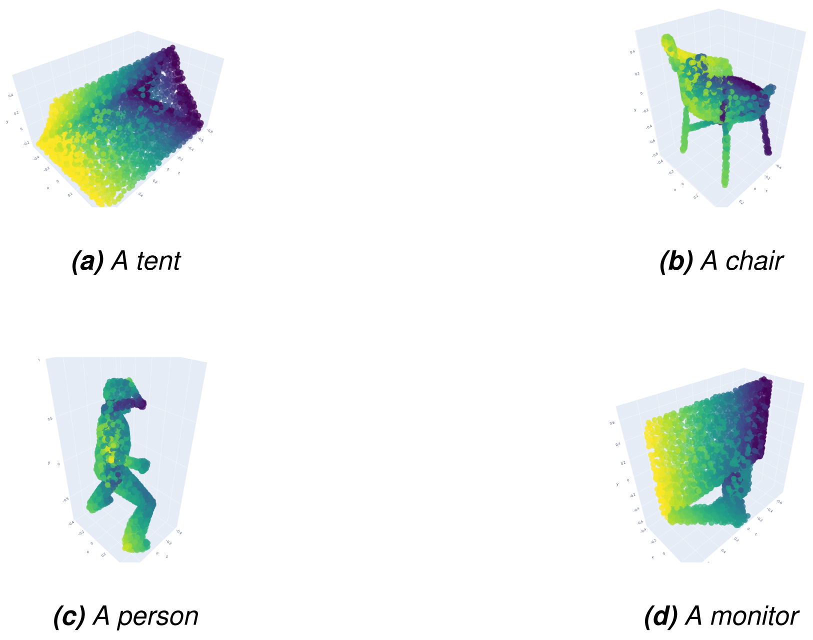
3.2 ModelNet40 dataset

ModelNet40 [16] is a publicly available large scale 3D CAD dataset used for benchmarking in 3D object classification and retrieval tasks developed by Princeton. It has 12,311 CAD models from 40 common object categories and is split into 9,843 objects for training and 2,468 objects for testing. Both the orientation and facing directions of the objects are aligned in the dataset. All classes are presented in Table 3.1.

airplane	bottle	cup	flower_pot	laptop	piano	sofa	toilet
bathtub	bowl	curtain	glass_box	mantel	plant	stairs	tv_stand
bed	car	desk	guitar	monitor	radio	stool	vase
bench	chair	door	keyboard	night_stand	range_hood	table	wardrobe
bookshelf	cone	dresser	lamp	person	sink	tent	xbox

Table 3.1. *ModelNet40 classes*

Since PointNet processes point cloud directly, a pre-processing step is needed to use ModelNet40 dataset. For each CAD object, we sample 2048 points on its surface area and normalize it into a unit sphere.



(a) *A tent* ***(b)*** *A chair*

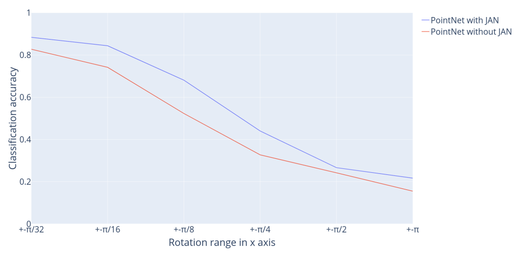
(c) *A person* ***(d)*** *A monitor*

Figure 3.2. *Examples of point cloud*

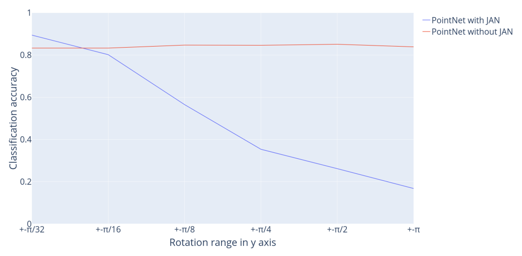
3.3 Experiment results

We first train PointNet with and without the Joint Alignment Network, in the same environment and with the same hyperparameter. After that, we test each model with the algorithm described in Listing 3.1 with 1000 test samples, and 6 different rotation ranges and calculate its accuracy.

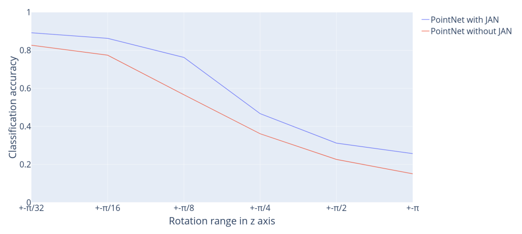
We can see that the performance of both models against rotation in x, z, and arbitrary axes are similarly poor. It is because the data used for training was only augmented with rotation in y-axis, so both models don't see any data rotated in other axes. Regarding



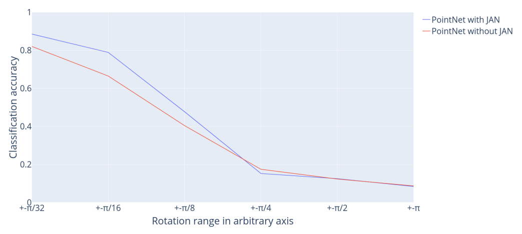
***(a)** Test data is rotated in x axis*



***(b)** Test data is rotated in y axis*



***(c)** Test data is rotated in z axis*



***(d)** Test data is rotated in arbitrary axis*

***Figure 3.3.** Comparision between PointNet in the presence and absence of Joint Alignment Network' robustness against rotation result (training data is augmented with y rotation)*

rotation in y-axis, PointNet without Joint Alignment Network's performance doesn't drop in all rotation ranges, but actually increases. This might be because the distribution of the original pose is much less compared to the distribution of the rotated poses in the training data, which makes PointNet without Joint Alignment Network performs better when we rotate the original test data before feeding it to the network. However, PointNet with Joint Alignment Network's performance drops rapidly as the rotation range widens. This suggests that the Joint Alignment Network does not help PointNet to be rotation invariant, but actually decreases its robustness against rotation transformation.

3.4 On data augmentation

Data augmentation has been known as one of the most effective way to increase deep learning models' robustness and decrease over-fitting [18]. Current data augmentation methods used in Pointnet (rotation about y-axis and jitter) are not enough to make PointNet geometric invariant. The reason that Qi et al [9] chose to augment data by rotating input point cloud around y-axis instead of arbitrary axis is because the test set shape is upright. Adding data augmentation that is not representative to test set data distribution can reduce the accuracy of the model. For better understanding of the relation between robustness and accuracy from data augmentation approach in PointNet, we design an experiment:

1. We replace the rotating about y-axis data augmentation option by rotating about arbitrary axis with the probability of 0.5, and train both PointNet with and without

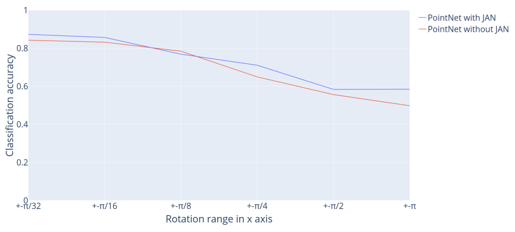
the Joint Alignment Network.

2. We then carry out the same rotation test (Listing 3.1) to both PointNet with and without the Joint Alignment Network.

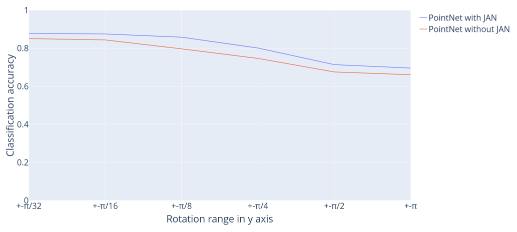
The models' accuracy on the original ModelNet40 test data comparison is shown on Table 3.2. There is a small difference between models' accuracy, but that could be the result of the non-determinism issue in Deep Learning [19]. The accuracy comparison between two models can be improved by training two models several times and calculating the average accuracy and variance. The comparision between PointNet in the presence and absence of Joint Alignment Network' robustness against rotation result (training data is augmented with arbitrary rotation) is described on Figure 3.4. We can see that when we augment training data with arbitrary rotation, PointNet with Joint Alignment Network performs slightly better than PointNet without the Joint Alignment Network. This suggests that with considerably more data, Joint Alignment Network can learn to transform the input data effectively. We also compare PointNets (training data is augmented with y rotation) and PointNets (training data is augmented with arbitrary rotation) as described in Figure 3.5 and Figure 3.6. Evidently, when training with arbitrary rotation augmentation, PointNets' robustness against rotation increase significantly.

Data augmentation methods	Model	Accuracy
Rotating about y axis	PointNet (with JAN)	89.7 %
	PointNet (without JAN)	83 %
Rotating about arbitrary axis (p = 0.5)	PointNet (with JAN)	87.9 %
	PointNet (without JAN)	86.5 %

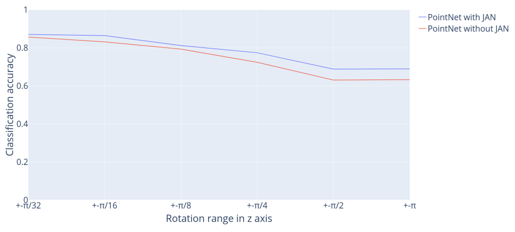
Table 3.2. *Classification results on the original ModelNet40 test data (no rotation)*



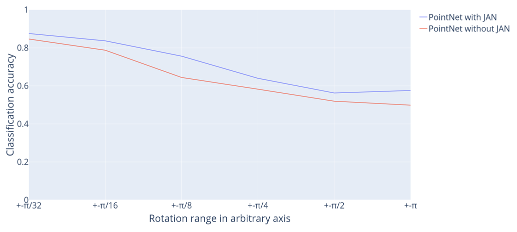
***(a)** Test data is rotated in x axis*



***(b)** Test data is rotated in y axis*

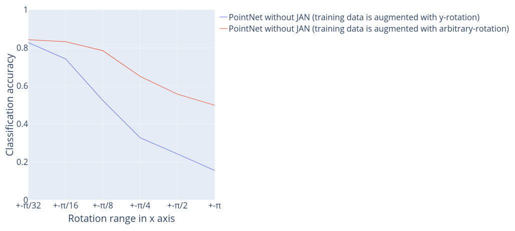


***(c)** Test data is rotated in z axis*

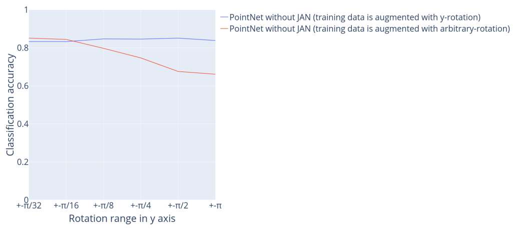


***(d)** Test data is rotated in arbitrary axis*

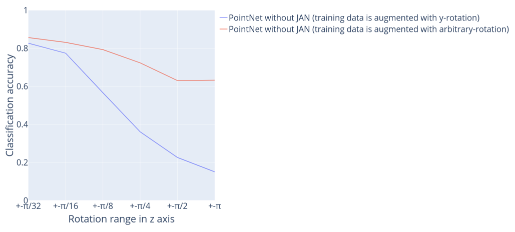
***Figure 3.4.** Comparision between PointNet in the presence and absence of Joint Alignment Network' robustness against rotation result (training data is augmented with arbitrary rotation)*



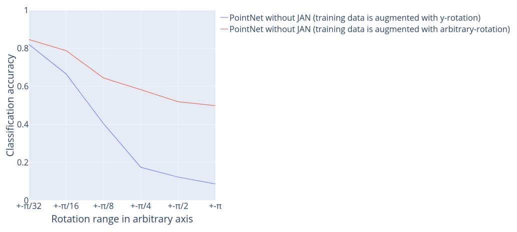
***(a)** Test data is rotated in x axis*



***(b)** Test data is rotated in y axis*

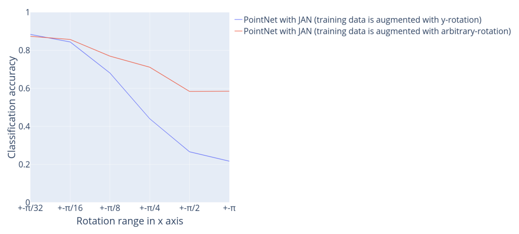


***(c)** Test data is rotated in z axis*

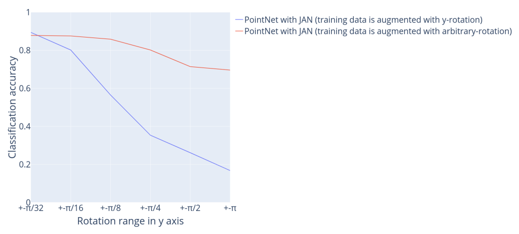


***(d)** Test data is rotated in arbitrary axis*

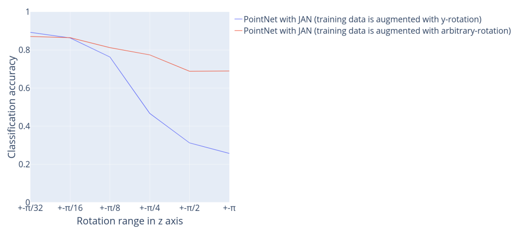
***Figure 3.5.** Comparision between PointNet without Joint Alignment Network (training data is augmented with y rotation) and PointNet without Joint Alignment Network (training data is augmented with arbitrary rotation)' robustness against rotation result*



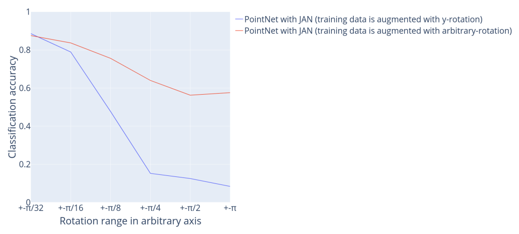
(a) *Test data is rotated in x axis*



(b) *Test data is rotated in y axis*



(c) *Test data is rotated in z axis*



(d) *Test data is rotated in arbitrary axis*

Figure 3.6. *Comparision between PointNet with Joint Alignment Network (training data is augmented with y rotation) and PointNet with Joint Alignment Network (training data is augmented with arbitrary rotation)' robustness against rotation result*

4. CONCLUSION

In this thesis, we studied and experimented if Joint Alignment Network helps PointNet become geometric invariant. The proposed test was rotating the input point cloud randomly before running it through the network to see if the performance drops. We carried out the test on 3D object classfication task using Modelnet40 dataset.

The obtained results show that with the original data augmentation, Joint Alignment Network can decrease PointNet's robustness against rotation transformation. Using stronger data augmentation of rotation about an arbitrary axis, both PointNet with Joint Alignment Network and PointNet without Joint Alignment Network' robustness increase overall. However, PointNet with Joint Alignment Networks performs better than PointNet without Joint Alignment Network. This implies that the Joint Alignment Network might require much more data to learn how to transform the data effectively. One possible solution to further analyze the effect of the Joint Alignment Network is to calculate the distance between the transformed object by the Joint Alignment Network and the original object (before rotation). A detailed analysis will be left for future work.

REFERENCES

[1] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[2] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J. and Wojna, Z. Rethinking the Inception Architecture for Computer Vision. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition,* 2016. URL: http://arxiv.org/abs/1512.00567.

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[4] Redmon, J., Divvala, S., Girshick, R. and Farhadi, A. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[5] Redmon, J. and Farhadi, A. *YOLO9000: Better, Faster, Stronger*. 2016. arXiv: 1612.08242 [cs.CV].

[6] Long, J., Shelhamer, E. and Darrell, T. *Fully Convolutional Networks for Semantic Segmentation*. 2015. arXiv: 1411.4038 [cs.CV].

[7] Ronneberger, O., Fischer, P. and Brox, T. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[8] Xiao, T., Liu, Y., Zhou, B., Jiang, Y. and Sun, J. *Unified Perceptual Parsing for Scene Understanding*. 2018. arXiv: 1807.10221 [cs.CV].

[9] Qi, C. R., Su, H., Mo, K. and Guibas, L. J. *PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation*. 2017. arXiv: 1612.00593 [cs.CV].

[10] Wikipedia contributors. *Symmetric function — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-April-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Symmetric_function&oldid=997796834.

[11] Wikipedia contributors. *Multilayer perceptron — Wikipedia, The Free Encyclopedia*. [Online; accessed 31-May-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Multilayer_perceptron&oldid=1013769694.

[12] Jaderberg, M., Simonyan, K., Zisserman, A. and Kavukcuoglu, K. *Spatial Transformer Networks*. 2016. arXiv: 1506.02025 [cs.CV].

[13] Qi, C. R., Yi, L., Su, H. and Guibas, L. J. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *arXiv preprint arXiv:1706.02413* (2017).

[14] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Jia, Y., Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[15] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[16] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X. and Xiao, J. *3D ShapeNets: A Deep Representation for Volumetric Shapes*. 2015. arXiv: 1406.5670 [cs.CV].

[17] Wikipedia contributors. *Euler's rotation theorem — Wikipedia, The Free Encyclopedia*. 2021. URL: https://en.wikipedia.org/w/index.php?title=Euler%27s_rotation_theorem&oldid=1014607746.

[18] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[19] Madhyastha, P. and Jain, R. *On Model Stability as a Function of Random Seed*. 2019. arXiv: 1909.10447 [cs.LG].