Neural Network for 3D Object Classification

Surabhi Gupta (2019701024), Ekta Gavas (2019701005)

***Abstract*—Object classification is one of the most common tasks in the field of computer vision nowadays. High-quality 3D object classification is an important component of many vision and robotics systems. We tackle the 3D object classification problem using two data representations viz basic 0/1 and complex representation, to achieve benchmark results on the Princeton ModelNet dataset.**

***Index Terms*—3D object classification, convolutional neural network, Voxnet, 3D SPN etc.**

I. Introduction

WITH the recent boost of inexpensive depth sensors, more and more 3D world is being modelled and hence the availability of 3D CAD model datasets. Three dimensional model classification is an important problem, with applications including self driving cars and augmented reality, among others.

3D content creation has been picking up pace in the recent past and the amount of information in the form of 3D models becoming publicly available is steadily increasing. This is good news for methods based on Convolutional Neural Networks (CNNs) whose performance currently rely heavily on the availability of large amount of data. A lot of methods based on CNNs have been proposed recently, for the purposes of image and object classification. They achieve significantly better results compared to using standard machine learning tools on top of hand crafted features.

In this report, we specifically focus on the classification task of 3D objects obtained from CAD models. Most of the recent systems which achieve state-of-the-art performance in 3D object classification on the ModelNet40 benchmark are based on CNNs. Volumetric representation in the form of binary voxels has been shown to be useful for classification and retrieval of 3D models. They train their network generatively. Then the introduction of Voxnet [1], a 3D CNN for 3D point cloud data which performed significantly better than other traditional methods.

The two representations are:

- ***Volumetric representation:** the 3D object is discretized spatially as binary voxels - 1 if the voxel is occupied and 0 otherwise.*
- ***Complex representation:** the 3D object is represented by a distance based transform on the binary voxel cubes, where under the same voxelization, for each voxel, we assign the squared minimum distance to the surface of the 3D object.*

II. Problem statement

We have to classify the 3D voxel based objects, like the ModelNet dataset using deep neural networks, along with dealing the problem of rotation invariance in 3D object classification.

III. Dataset

We are using ModelNet dataset from Princeton University for this problem. The 40-class ModelNet40 contains CAD models from 40 categories that are used to train the deep network. A subset of ModelNet dataset is ModelNet10 which has 10 categories.

A. ModelNet40

ModelNet40 contains 9449 shapes covering 40 common categories which are airplane, bathtub, bed, bench, bookshelf, bottle, bowl, car, chair, cone, cup, curtain, desk, door, dresser, flower pot, glass box, guitar, keyboard, lamp, laptop, mantel, monitor, night stand, person, piano, plant, radio, range hood, sink, sofa, stairs, stool, table, tent, toilet, tv stand, vase, wardrobe, xbox. The dataset is divided into 7589 train + 1860 test samples.

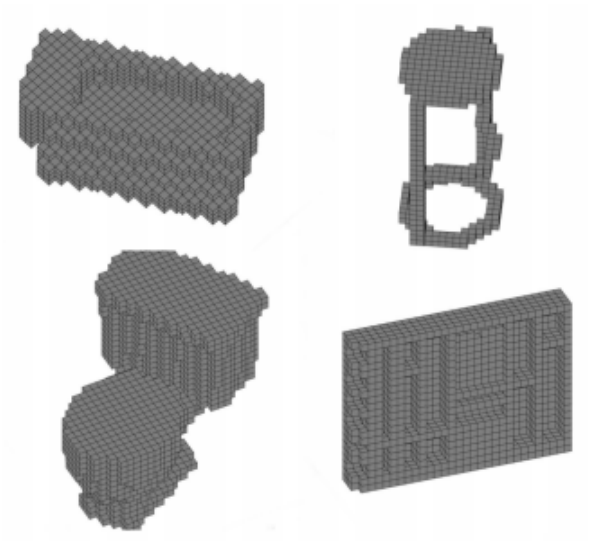

Fig. 1. Sample voxelized version of 3D CAD models from ModelNet40. Top-left: bathtub, top-right: stool, bottom-left: toilet, bottom-right: wardrobe.

B. ModelNet10

ModelNet10 contains 4899 shapes covering 10 classes which are bathtub, bed, chair, desk, dresser, monitor, night stand, sofa, table, toilet. The dataset is divided into 3991 train + 908 test samples.

C. Data preparation

The dataset contains files in .OFF format and samples in each class are split into train and test. A utility 'binvox' is used to read the .OFF files and create corresponding .binvox format files, which can then be read in python by its corresponding python module.
We have split the training samples into training and validation data with a split of 30 %. The class distributions for ModelNet10 and ModelNet40 for training, validation and test sets are shown in Fig 2 and Fig 3 respectively. Here, we can see that there is an imbalance in number of samples of classes in ModelNet40 dataset.

IV. Technical Approach

A. 0/1 representation for voxels

'Voxel' stands for 'volumetric element' or 'volumetric pixel' and is a three-dimensional element analogous to what pixels are in two-dimensions. The initial point-cloud data can be converted to a voxel structure, denoted as Vd(x, y, z), by imposing a lattice. The first step of the voxel conversion procedure is to define an N x N x N grid box in such a way that the barycenter of the point cloud coincides with the center of the box. Then, we define a binary voxel occupancy function V(x, y, z) on this grid. This is simply an indicator function: If, in a cell at location (x, y, z), there do not exist any points of the cloud, V(x, y, z) is set to zero. If there are one or more points in that cell, then the binary function at that voxel location assumes the value of one. Therefore, all cells on the face have the value of one, and the rest of the cells in the space are set to zero, which, in effect, defines a 3-D shell.

We test the VoxNet [1] as our baseline model. The architecture of VoxNet is shown below:

Layers	Parameters
fully connect	40
drop3	p = 0.4
fully connect	128*10
drop2	p = 0.4
pool2	pool shape [2 2 2]
conv2	receptive field 3x3x3 filterNum 32
drop1	p = 0.2
Conv1	receptive field 5x5x5 filterNum 32
Input Layer	Size 32 x 32 x32

Fig. 4. Architecture of VoxNet.

B. Complex representation

It is advantageous to convert the binary voxel data into a continuous form via the distance transformation. We apply 3-D distance transform to the binary function V(x, y, z) to fill the voxel grid and obtain Vd(x, y, z). The distance trans-form is defined as the smallest euclidean distance of a voxel point to the binary surface. This function gets a value of zero on the face surface, and it increases as we get further away from the surface. By using the distance transform, we distribute the shape information of the surface throughout the 3-D space and obtain a smoother representation compared to the binary voxel.

C. 3D Spatial Transformer networks

Spatial transformer networks were proposed by [2] to deal with the problem of rotational invariance.The basic idea is to consider the spatial transform as regression problem as we can specify the transformation parameters. The architecture is shown in Fig. 6. This idea was adopted and a 3D spatial transform network was built into the VoxNet. The basic architecture is shown in Fig. 5

Fig. 5. STN architecture

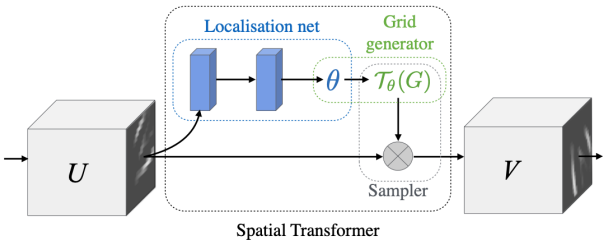

Fig. 6. Spatial transformer network

SPN module consists of three submodules. First one is the localization network which regresses the affine transformation matrix theta such that it can predict correct labels for classification. This matrix can account for different transformations like rotation, zooming etc. The localization network takes the input feature map $U \epsilon\ R^{H \times W \times L}$, where $H \times W \times$ L is the size of the voxel data and outputs θ, the parameters of the transformation T_θ to be applied to the feature map: $T_\theta = f_{loc}(U)$. θ is transformation matrix of shape 3 x 2 in 2D case and 3 x 4 in 3D. Transformation matrix for 3D is shown in Fig. 7.

$$\mathcal{T}_\theta = \begin{pmatrix} \theta_1 & \theta_{12} & 0 & \theta_{14} \\ \theta_{21} & \theta_{22} & 0 & \theta_{24} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Fig. 7. Transformation theta

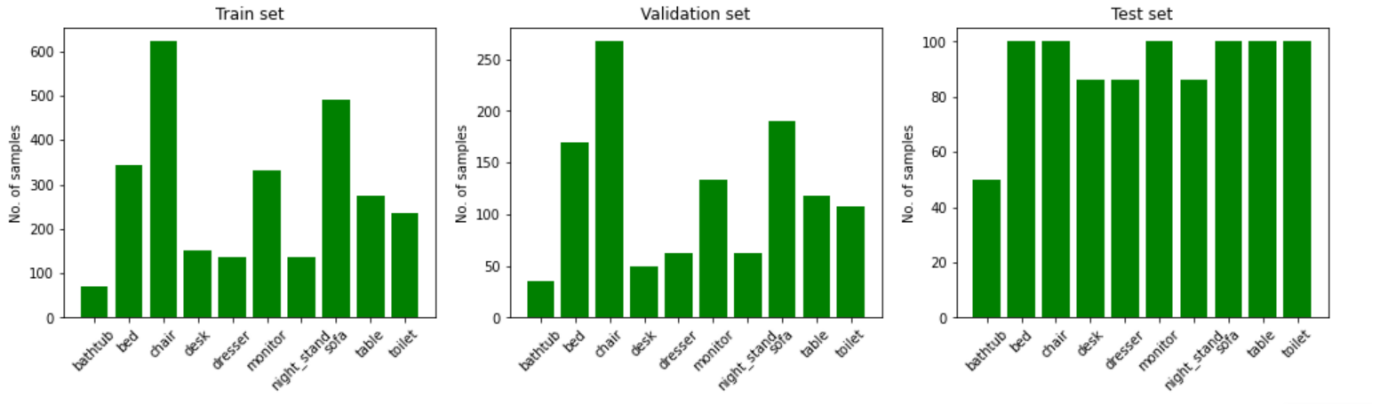

Fig. 2. Class distribution for ModelNet10

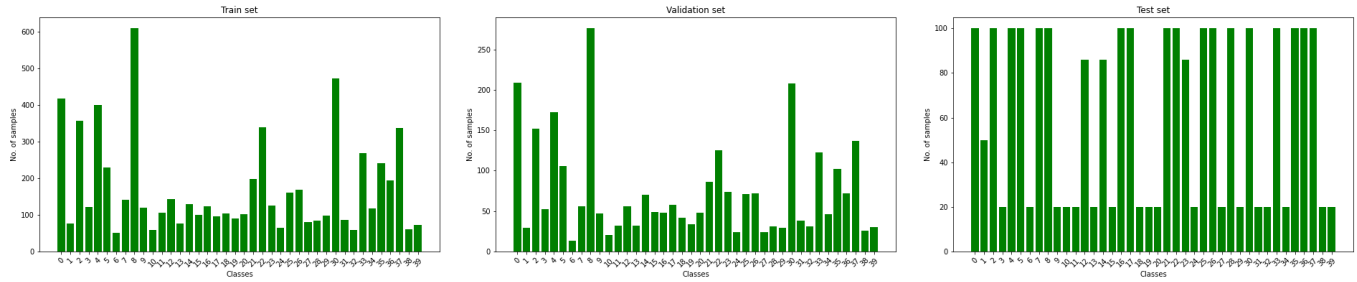

Fig. 3. Class distribution for ModelNet40

$\theta_{11}, \theta_{12}, \theta_{21}, \theta_{22}$ account for rotation/scaling and θ_{14}, θ_{24} are for translation. Also, other parameters are zero as we consider rotation only around the gravity direction. The localization network can be any network. In the paper [3], authors are using same VoxNet architecture with 6 output units instead, for the localization network to regress the 6 unknown parameters of T_θ. Due to the need to convert the 6 parameters in the shown 3 x 4 matrix format for applying the transformation to 3D object later on, we changed this proposed architecture so that the localization network regresses all 12 parameters instead of 6. We then make the other parameters to 0 or 1. This will also ensure differentiability. The basic architecture for localization network is still VoxNet.

The second part of 3D-SPN is the grid generator. Rather than applying the transformation directly to the initial input, a sampling meshgrid of the same size as the initial voxel input U is generated. A meshgrid is a set of indices (x_i, y_i, z_i) that cover the whole voxel input space. Specifically, for each output voxel (x_i^t, y_i^t, z_i^t) must be on a regular grid G. Then given T_θ, we can find the position of the source voxels (x_i^s, y_i^s, z_i^s) by matrix multiplication of theta by coordinates (x_i^t, y_i^t, z_i^t) and a bias term (1).

$$\begin{pmatrix} x_i^s \\ y_i^s \\ z_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \begin{pmatrix} \theta_1 & \theta_{12} & \theta_{13} & \theta_{14} \\ \theta_{21} & \theta_{22} & \theta_{23} & \theta_{24} \\ \theta_{31} & \theta_{32} & \theta_{33} & \theta_{34} \end{pmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ z_i^t \\ 1 \end{pmatrix} \quad (1)$$

where (x_i^t, y_i^t, z_i^t) is the target position of the output voxel data and (x_i^s, y_i^s, z_i^s) is the source position of the input voxel data. And we used normalized coordinates such that $-1 \leq (x_i^t, y_i^t, z_i^t) \leq 1$ when within the spatial bounds of the output and (x_i^s, y_i^s, z_i^s) when within the spatial bounds of the input.

In the third part, the sampled output V using the initial feature map, the transformed grid and a differentiable interpolation function, here bilinear, is produced. Interpolation is required as we need to map the result of sampling (fractions of pixels) to pixel values (integers). Fig. 8 shows the sampling and interpolation for 2D case where input has C channels. Similarly, we can have a sampling kernel for 3D as in shown in equation (2) where V_i is the voxel value at $(x_i^t, y_i^t, z_i^t), \psi_x, \psi_y and \psi_z$ are the parameters of a generic sampling kernel k()

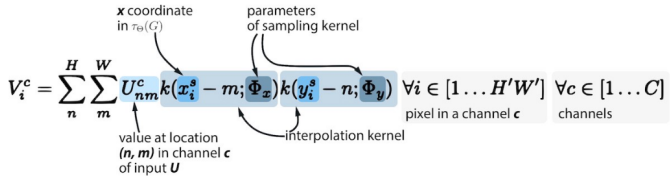

Fig. 8. Sampling and interpolation

$$V_i = \sum_n^H \sum_m^W \sum_l^L U_{nml} k(x_i^s - m; \Psi_x) k(y_i^s - n; \Psi_y) k(z_i^s - l; \Psi_z) \tag{2}$$

Specifically we use a bilinear sampling kernel, which gives equation (3)

$$V_i = \sum_n^H \sum_m^W \sum_l^L U_{nml} \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|) \max(0, 1 - |z_i^s - l|). \tag{3}$$

In this scheme, we can derive the backpropagation of the loss by computing the gradients:

$$\frac{\partial V_i}{\partial U_{nml}} = \sum_n^H \sum_m^W \sum_l^L \max(0, 1 - |x_i^s - m|) \max(0, 1 - |y_i^s - n|) \max(0, 1 - |z_i^s - l|)$$

$$\frac{\partial V_i}{\partial x_i^s} = \sum_n^H \sum_m^W \sum_l^L \left[\max(0, 1 - |y_i^s - n|) \max(0, 1 - |z_i^s - l|) \cdot \begin{cases} 0 & |m - x_i^s| \geq 1 \\ 1 & x_i^s \leq m < x_i^s + 1 \\ -1 & x_i^s - 1 < m \leq x_i^s \end{cases} \right].$$

TABLE I
CLASSIFICATION ACCURACY FOR MODELNET10

Algorithm	Training acc	Validation acc	Testing acc
Modelnet10+Baseline	91.55	89.01	**85.5**
Modelnet10+Complex	**94.13**	**89.06**	83.4
Modelnet10+3D SPN	90	88	54.4

TABLE II
CLASSIFICATION ACCURACY FOR MODELNET40

Algorithm	Training acc	Validation acc	Testing acc
Modelnet40+Baseline	80	**77.6**	**77.5**
Modelnet40+Complex	**81.36**	75.66	72.97
Modelnet40+3D SPN	75.6	76	53.4

D. Training and Classification

The original paper[3] proposed augmenting data by rotating the models every 30 degress and test data consist of two sets, one augmented with 30 degree rotation and other with 60 degree. We, instead, augment the training data by randomly rotating every sample. This would help in better generalization of the model and may also avoid overfitting caused due to such augmentation. Similar to the original paper[3], all rotations are performed along the gravity direction only. The objects are rotated by a random angle in [0,360). For simplicity, rotating angles are taken as whole numbers rather than decimal values. We augment only the train set and not validation and test set.

We use the Cross Entropy loss and the weights are updated using Adam optimizer with momentum and weight decay. The best model is saved during training and at the test time, we evaluate the model against test set.

V. EXPERIMENTAL RESULTS

A. Visualization

After training is completed, we take the intermediate output from 3D SPN and plot the generated transformations. We then compare these learnt voxelizations with the original voxelizations to understand how the 3D SPN works. We note that the SPN tries to align the randomly rotated objects along a single axis. Original and tranformed voxelized grids are shown in Fig 9 and Fig 10.

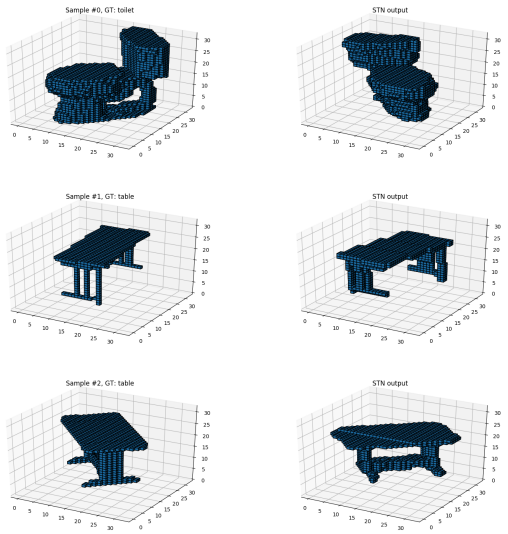

Fig. 9. Output for 3D SPN in ModelNet10

B. Evaluation metrics

To evaluate our model, we used three different performance metrics: Accuracy, Precision, Recall and F1-score.

1) Accuracy: Tables III and IV shows the classification accuracy for 3D-SPN on both the datasets.

The performance of the complex representation suggests that it might be a good choice for representing 3D voxel based data for the classification task. The distribution of the voxel values over the whole voxel cubic is approximately rotation invariant because it relied on the relative distance to the object itself. The comparison of the baseline Voxnet with different representations shown in Table 1 shows that the distance

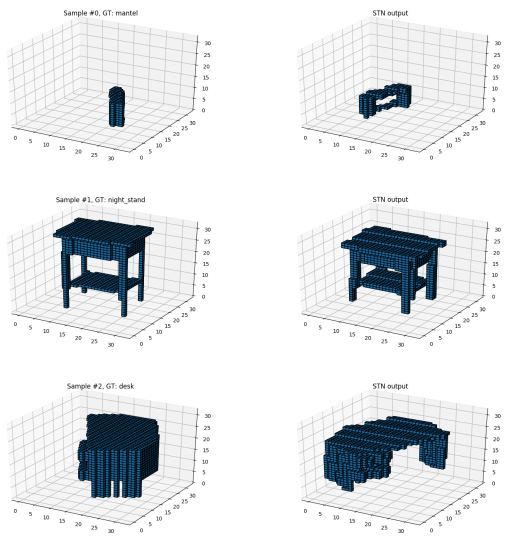

Fig. 10. Output for 3D SPN in ModelNet40

based repsresentation is on par with normal 0/1 representation.

2) Precision: Precision can be calculated as:

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives}$$

3) Recall: Recall can be calculated as:

$$Recall = \frac{TruePositives}{TruePositives + FalsePositives}$$

4) F1 score: F1 score can be calculated as:

$$F1score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

TABLE III
PRECISION, RECALL AND F1-SCORE FOR COMPLEX+MODELNET10

Class	Precision	Recall	F1 Score	Support
0	0.9	0.54	0.675	50
1	0.847	0.94	0.891	100
2	0.809	0.89	0.848	100
3	0.686	0.558	0.615	86
4	0.8	0.791	0.795	86
5	0.970	0.96	0.965	100
6	0.783	0.628	0.627	86
7	0.905	0.95	0.927	100
8	0.677	0.86	0.758	100
9	0.951	0.97	0.96	100
Avg/Total	0.833	0.830	0.826	908

TABLE IV
CLASSIFICATION ACCURACY FOR 3D SPN+MODELNET10

Class	Precision	Recall	F1 Score	Support
0	0.208	0.100	0.135	50
1	0.737	0.870	0.798	100
2	0.598	0.730	0.658	100
3	0.417	0.291	0.342	86
4	0.542	0.523	0.533	86
5	0.680	0.680	0.680	100
6	0.357	0.233	0.282	86
7	0.600	0.870	0.710	100
8	0.453	0.340	0.389	100
9	0.536	0.670	0.596	100
Avg/Total	0.533	0.563	0.539	908

VI. DISCUSSION AND SUMMARY

In this work we focus on the problem of 3D object classification using CNN architecture, VoxNet. To address the problem of rotation invariance and to improve the performance, we implemented another method of representing 3D data using distance transform. Another approach to deal with this problem was a 3D Spatial Transform Network.

We evaluated our network on ModelNet40 and its subset, ModelNet10. We observed that, of the two complex representation models relatively performed better. Due to imbalance in test and training data, the good precision was not obtained for all the classes. Also, batch normalization and reducing the architecture of 3D SPN etc techniques also didn't improve the model performance much. This may be due to the fact that the dataset contained many noisy samples as well.

Mixing all the dataset samples and again splitting may improve the performance. Efficiently detecting the noisy samples and removing them is also one possibility. We can also address the class-imbalance problem by augmentation or weighted sampling. There is a scope of further exploration in this direction.

The entire project has been developed using the Pytorch framework along with other common Python libraries. The code is available at: ***https://github.com/SURABHI-GUPTA/Smart_Visioners_Team-ID_2.git***

REFERENCES

[1] Maturana, Daniel and Scherer, Sebastian, *Voxnet: A 3d convolutional neural network for real-time object recognition*, 2015

[2] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, *Spatial Transformer Networks*, 2015

[3] Ling Shao, Peng Xu, *Neural Network for 3D Object Classification*, 2017

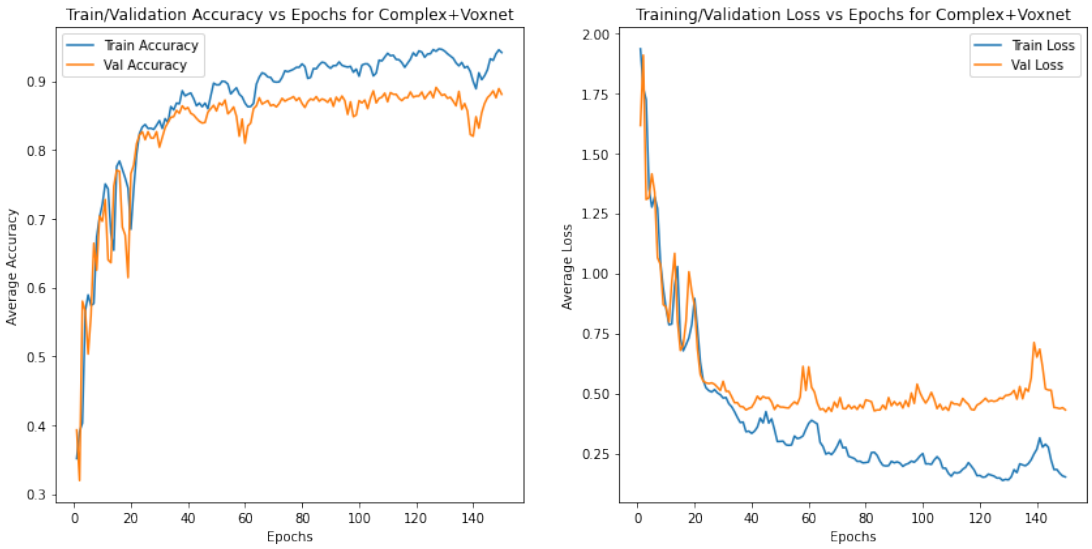

Fig. 11. Accuracy(left) and Loss(right) using Complex Representation on ModelNet10

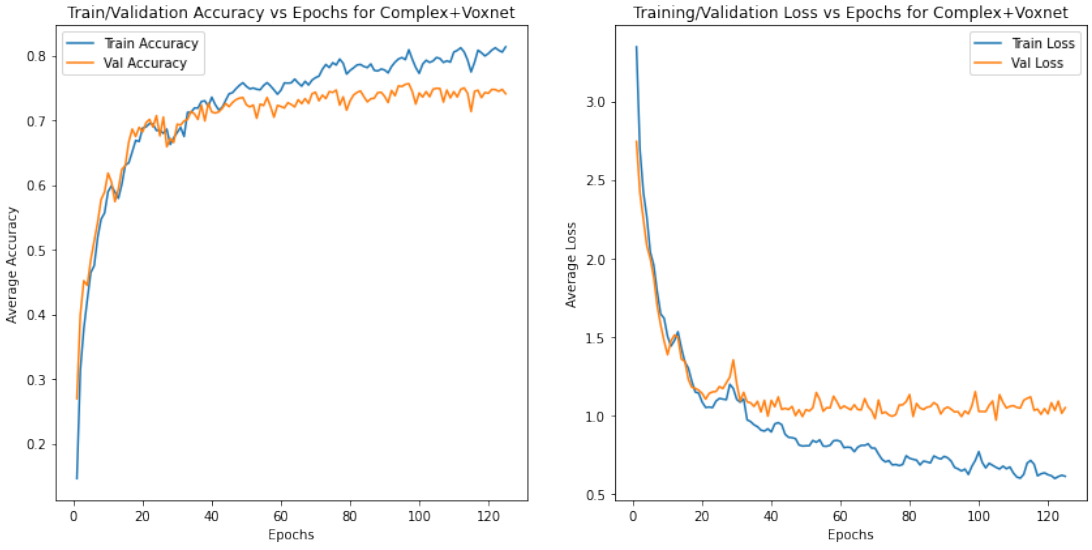

Fig. 12. Accuracy(left) and Loss(right) using Complex Representation on ModelNet40

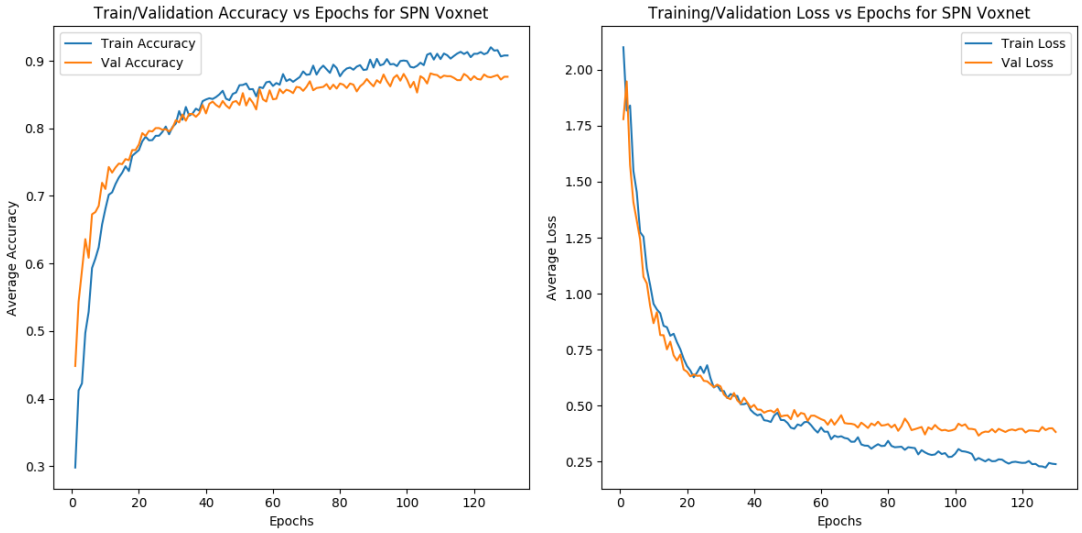

Fig. 13. Accuracy(left) and Loss(right) using 3D SPN on ModelNet10

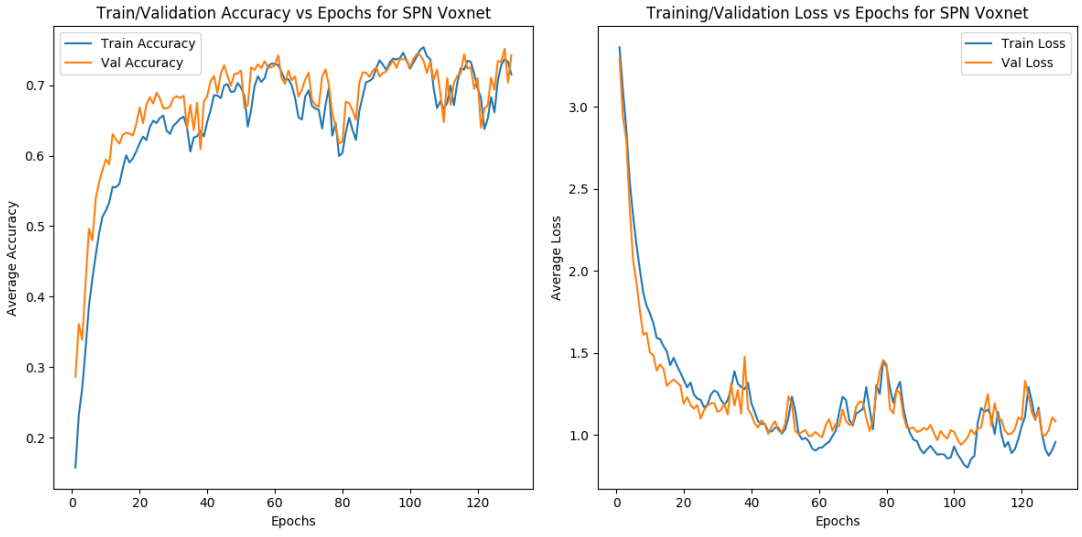

Fig. 14. Accuracy(left) and Loss(right) using 3D SPN on ModelNet40