Object Recognition for Point Cloud using Neural Networks

EECS 545 Final Project Report

December 17, 2017

Mrunmayee Deshpande[†]*, Lu Gan[†]*, Bruce JK Huang[†]*, and Abhishek Venkataraman[†]*
{msdesh, ganlu, bjhuang, abhven}@umich.edu

***Abstract*— Object recognition is critical in cluttered indoor environments for successful operation of robots. While neural networks have shown to perform well on images, there performance on 3D data is lesser studied. We explore the possibility of fine-tuning the existing neural network approaches to study the performance. We also additionally evaluate if the performance can be improved by replacing the last layer of the neural network with a SVM classifier.**

I. Introduction

Semantic object recognition is a critical problem for robots and autonomous vehicles to be able to operate in clutterd indoor environment. Point clouds are commonly used for perceiving objects and scenes in a 3D setting. Most recently, active range sensors such as LiDARs and RGBD cameras make them even more reliable and accurate. They are also already used in many applications such as modeling tools and autonomous systems, including cars [18], quadrotors [11] and helicopters [5]. Before the Deep Learning [13], most of the state-of-the-art use a traditional pipeline: extraction and aggregation of hand-engineered feature, which then are fed into an classifier such as SVMs. Until more recently, a lot of early work has been largely superseded by approaches based on Deep Learning. These two years, more and mroe papers focused on semantic segmentation of objects in a scene using Deep Learning. In fact, semantic segmentation in 2D images is a well studied problem, with challenges such as PASCAL VOC [8] have shown several successful architectures. However the same problem in 3D space is much less studied. The availability of low cost structured light sensor, has surged the interest in perceiving scene in 3D space. Currently a lot of work focus on using depth information for building occupancy map, spatial filtering, etc. However, there is little work on using depth information for end to end object classification.

Fig. 1: Fetch Robot manipulating warehouse objects[1]

Support Vector Machines (SVMs) have been popular in 3D object recognition [15] for quite some time, but their accuracy tend to saturate. We found some recent work based on neural network in this area, as discussed in Section II to show very promising results. We were interested in using the existing neural network based method and investigating if fine tuning the architecture enhances the results.

In this project, we focus mainly on using depth information to classify objects, specifically in indoor scenes. We first recreated one the architectures in tensorflow and then added additional layer and SVM classifies on to investigate if it increased the classification accuracy. The details of our implementation are explained in Section III. We then finally compare our classification accuracy with the existing baseline in Section IV.

[†] Robotics Institute, University of Michigan.
* Equal contribution by each of the authors.

[1] http://fetchrobotics.com/

II. Related Work

In this project, we focus on ModelNet10 [20]. It is remarkable that the training and test splits are defined by the dataset itself and also it is worth saying that the number of examples per class are not balanced, as seen in Table I. There are many state-of-the-art methods listed in the leaderboard, listed in Table II. We took the VoxNet [14] proposed by Maturana and Scherer since their result was good baseline and would give us margin for improvement. The original work by Wu et al. [20] introduced the ModelNet dataset and a Convolutional Deep Belief Network (CDBN) to represent and learn 3D shapes as probability distributions of binary variables on volumetric voxel grids, where they reached 83.50% accuracy. Their method set a baseline and has been used to benchmark various 3D object recognition methods.

Class Number	Category	Training Set	Testing set
1	Desk	200	86
2	Table	392	100
3	Nighstand	200	86
4	Bed	515	100
5	Toilet	344	100
6	Dresser	200	100
7	Bathtub	106	50
8	Sofa	680	100
9	Monitor	465	100
10	Chair	889	100

TABLE I: ModelNet-10 examples per class distribution

As followed is the DeepPano [16], raising the accuracy upto 85.45% by changing 3D shapes to panoramic views, using a cylinder projection around their principle axes, and learning them with a Convolutional Neural Networks specifically designed for that purpose. Wu et al. [19] proposed the 3D Generative Adversarial Network (GAN), which can generate or sample 3D objects. They proved to work well in the classification problem and their model achieved accuracies up to 91%.

The architecture we used was built on top of the method proposed by Maturana and Scherer, Voxenet [14]. They showed that put a pure 3D Convolutional neural networks together with a volumetric occupancy grid representation would be helpful to recognize 3D shapes efficiently. Their model, achieved a 92.00% on the benchmark. Though there have been more recent work after this, we treated this paper as baseline since it give us room for improvement while exploring various architectures.

The following is the 2.5D approach, proposed by Bai et al. [2] where they used GPU acceleration and Inverted File Twice (GIFT): a real-time matching method that combines projective images of 3D shapes with a CNN so as to extract features that are later matched and ranked to provide a candidate list. By using this approach, they got a slightly better result, 0.35%, than VoxNet [14] reaching a 92.35% accuracy. The method proposed by Johns et al. [12]: they proved that multiview image sequences could boost the accuracy to 92.80%; they used a CNN to independently classify image pairs from sequences, and then classify them again weighting the contribution of each pair.

FusionNet [10] combines both approaches: volumetric representations (binary voxel grids) and pixel representations (projected images); they use both representations to feed two volumetric CNNs and a MultiView Convolutional Neural Network (MVCNN), as a result they get a 93.11% accuracy. The MVCNN approach was already introduced by Su et al. [17] which used a Convolutional neural networks to learn to classify objects using a collection of rendered views for each one; however, they did not report any result for the ModelNet10 challenge. The next significant jump was carried out by Sedaghat et al. who introduced the object orientation prediction, in addition to the class label itself, to boost classification accuracy; their ORION network is a 3D CNN which produces class labels and orientations as outputs and uses both of them to contribute to the training. By adding orientation estimation as an auxiliary task during training they were able to learn orientation invariance and lift accuracy up to 93.80%.

Method	ModelNet10 Accuracy	ModelNet40 Accuracy
VRN Ensemble	97.14%	95.54%
LonchaNet	94.37%	N/A
ORION	93.80%	N/A
FusionNet	93.11%	90.80%
Pairwise	92.80%	90.70%
GIFT	92.35%	83.10%
VoxNet	**92.00%**	**83.00%**
3D-GAN	91.00%	83.30%
DeepPano	85.45%	77.63%
3DShapeNets	83.50%	77.00%
MVCNN	N/A	90.10%

TABLE II: ModelNet leaderboard in November, 2016

The current state-of-the-art of the method is Voxception-ResNet (VRN) ensemble model, proposed by Brock et al. [4]. This model outperforms other methods by a large margin with 97.14% accuracy. The

VRN architecture is based on ResNet [9] but instead of using standard ResNet blocks, it uses inception blocks produced by concatenating bottleneck and standard ResNet blocks. A voxellized volumetric input is fed to an ensemble of those VRNs whose predictions are summed to generate the output.

‘

III. Methodology

A. Motivation for Our Approach

Previous work in the field of 3D semantic object detection makes use of point cloud data for feature extraction and classification. However, the ability of volumetric representation to distinguish free space from unknown space makes it richer than the point clouds. The volumetric representation in the form of occupancy grids makes it possible to incorporate sensor readings coming from different viewpoints at different times. Occupancy grids are 3D lattice of random variables which are compatible with simple and efficient data structures. Occupancy grids are more discriminative representation of data resulting into better performance of CNN for 3D structures. This section describes detailed implementation of convolutional neural networks on 3D occupancy grids, obtained by pre-processing of 3D point cloud data.

In contrast to deep neural networks with dense weights, *sparse kernel machines* such as Support Vector Machines (SVMs) and Relevance Vector Machines (RVMs) are methods which explore sparsity of the models. Sparse kernel machines are designed to deal with the limitation of many kernel methods that evaluating kernel function on every pair of the training data is computationally expensive and even infeasible for a large dataset [3]. For instance, Gaussian Processes (GPs) use *covariance function* of all training data as kernel function, resulting in high prediction computational complexity (cubic in the number of training data). Sparse kernel machines select only a small subset of training data during the process of learning, yielding a sparse solution which only depends on this subset. Therefore, both training and prediction processes of sparse kernel machines can be fast.

Support vector machine is one of the most popular sparse kernel methods in machine learning and computer vision, due to the sparsity and nice property of convex optimization. This section explores 3D object recognition using SVM, analyzes the effects of different multi-classification strategies, kernels and hyper–parameters to recognition accuracy, and finally compare the performance with recognition performance using 3D CNN described before.

B. Data Pipeline for Neural Network

The 3D point cloud data is obtained from active sensors such as RGB-D cameras and LiDARs. For semantic object detection, the point cloud data is segmented into bounding boxes, called as segments. Our task is to predict object class labels for the segments. For this task, the segments are further processed to generate 3D occupancy grids which are the inputs for our convolutional neural network.

Convolutional neural networks prove to be a very suitable option for this task as they can learn local spatial filters which is useful for the classification task. Also, the multiple levels in the network incorporates hierarchy of complex features which represents larger regions of space. This leads to generating a global label for the occupancy grid. The pipeline of our architecture of our network is explained below.

The Input Layer accepts a fixed size grid of 30x30x30 voxels. The convolutional layers accepts four dimensional input of which first three dimensions are spatial and fourth dimension is number of filters. The output from convolutional layer is passed through a non-linearity unit, Leaky ReLU with parameter 0.1. Pooling layers are used for down sampling the output from convolutional layers by a factor of 2. Fully connected (FC) layers are implemented with the activation functions ReLU and Softmax. FC layers are initialized with zero-mean Gaussian and = 0.01. Dropout layer is added after each layer. The output of this layer is a one hot vector containing the object class prediction. Our convolutional neural network is a 5 layers network as shown in Fig. 2.

C. Sparse Kernel Machine Object Recognition

Let a set of input-target pairs $\mathscr{D} \triangleq \{x_i, y_i\}_{i=1}^{n_t}$ be the training set where $x_i \in \mathbb{R}^{n_d}$, $y_i \in \{1,-1\}$, n_d and n_t are the dimension and number of inputs, respectively. A soft kernel SVM takes the following form:

$$\begin{aligned} &\min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n_t} \zeta_i \\ &\text{subject to } y_i(w^T \phi(x_i) + b) \geq 1 - \zeta_i, \\ &\qquad\qquad \zeta_i \geq 0, i = 1, ..., n_t \end{aligned} \tag{1}$$

with the classification boundary as:

$$\text{sign}\left(w^T \phi(x) + b\right) \tag{2}$$

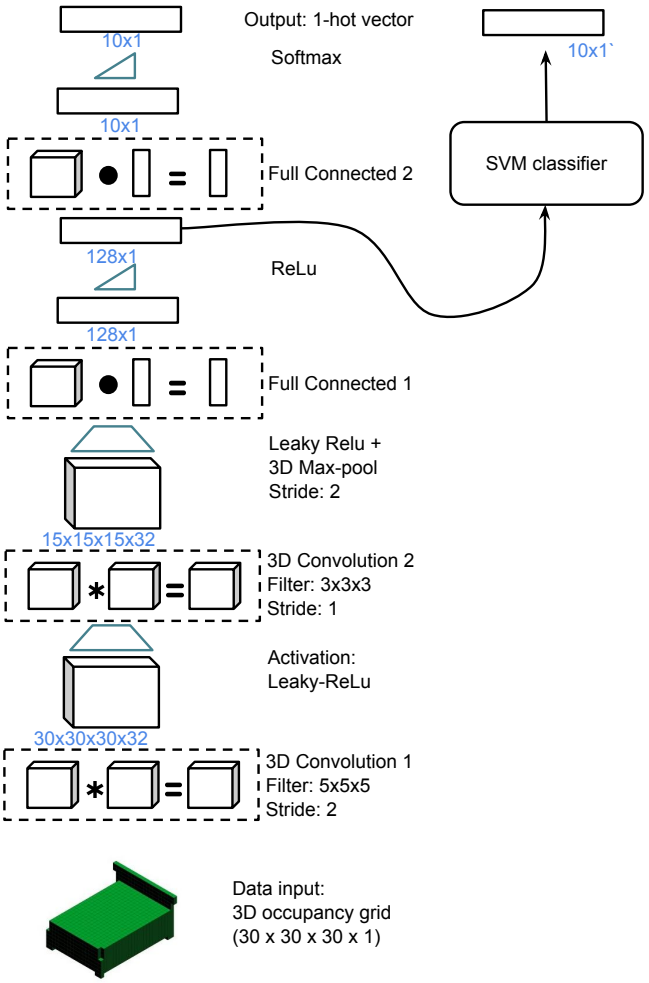


Fig. 2: Baseline Architecture - Voxnet. Features computed from FC 1 layer is used to train SVM classifier.

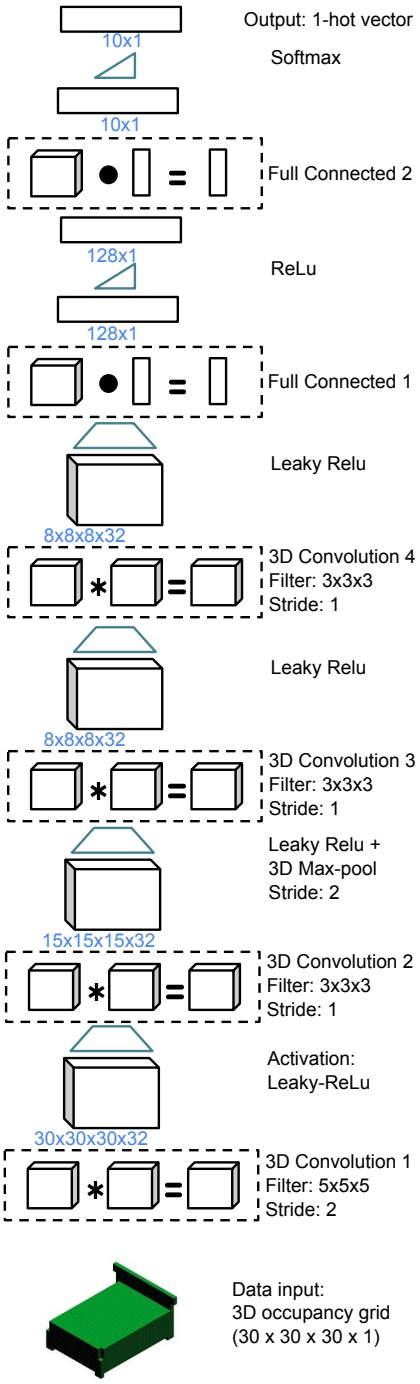


Fig. 3: Our architecture with SVM classifier

where $C > 0$ controls the penalty of errors $\zeta_i, i = 1,...,n$, $w \triangleq \text{vec}(w_1,\ldots,w_{n_b})$ is the parameter vector (weights) of a set of n_b basis functions $\phi(x) \triangleq \text{vec}(\phi_1(x),\ldots,\phi_{n_b}(x))$. The basis function implicitly map the input vector into a higher dimensional space, and can be specified by a kernel function $k(x_i,x_j) \triangleq \phi(x_i)^T\phi(x_j)$. The objective of SVM is to learn the parameter vector w and intercept b such that the model generalizes well to new unseen inputs x_* (test data).

We formulate 3D object recognition as a multi-class classification problem, and solve it using SVM binary classifier and several multi-classification strategies, such as one-vs.-one and one-vs.-rest. As neural networks can be regarded as learning feature detectors automatically from data, features extracted from NN are usually more effective and discriminative than hand-designed features. Therefore, we decided to use our trained 3D CNN model as feature extractor, to transform input data directly to feature vectors by running a forward prediction on 3D CNN except the last fully-connected layer. The motivations here are two-folds. First, we think SVM will outperform a fully-connected layer due to the power of non-linear kernels; Second, the outputs of SVM are sparse, and training and prediction can be fast.

The sparse kernel machine object recognition problem can be formulated as follows: Let $\mathscr{X} \subset \mathbb{R}_d^n$ be the set of input features, and $\mathscr{C} = \{c_k\}_{k=1}^{n_c}$ be the set of object class labels. The problem is that given the set of training data $\mathscr{D} \triangleq \{x_i, y_i\}_{i=1}^{n_t}$ whose elements take values $x_i \in \mathscr{X}$ and $y_i \in \mathscr{C}$, we wish to explore the correlations between input features and object class labels and estimate the class label for new data x_*.

Multi-class classification can be solved by training multiple SVM binary classifiers to learn the decision boundary for each object class, and the final decision is made by integrating decisions from all classifiers. The *one-vs.-rest* strategy builds classifier to separate one object class from all other classes, whereas *one-vs.-one* approach constructs $n_c(n_c-1)/2$ classifiers where each one constructs a boundary for two of all object classes. We use both strategies in experiments and the results are given in Section IV.

IV. Evaluation and Experiments

A. Dataset Selection

We chose ModelNet10 dataset for comparing performance between the baseline and our methods. ModelNet10 is a subset of the popular ModelNet40 dataset, which has similar classes as the NYUv2 dataset. ModelNet40 has 40 object categories while ModelNet10 has 10 object categories. We use voxelized version of the dataset which is scaled to fit 30x30x30 grid. We train and evaluate our model for ten categories mentioned in Table I.

B. Neural Network Methods

For the Theano implementation, we use the voxelized version of ModelNet 10, which is included in the source code distribution. However, it comes with .mat files, so we need to convert them into .npy files. Since the codes online are not always working at the first time, We then tweak around to make sure we can get the same results as the author had. We then trained it from scratch; the training loss and training accuracy is shown in Fig. 4. For Theano, the visualization is not really well implemented. We saved the learning curves in a html file format so that we could visualize it at the localhost in real-time. We also found some miss-classified labels where we will be talking in Section V.

To evaluate our proposed methods, we first implemented the baseline network in tensorflow. We used similar training parameters as above, stochastic gradient descent with momentum. The learning rate used was 0.001 and momentum factor, 0.9. We trained and evaluated the network on ModelNet dataset. The architecture is detailed in Figure 2. As you can see, the neural network part is identical to the Voxnet architecture with two 3D convolution layer followed by two fully connected (FC) layer. After training the baseline network, we removed the last FC layer and used the latent representation at end of FC layer 1 as input to the SVM classifier. Details of the our SVM implementation is explained in the subsequent subsections.

We also investigated if increasing the number of layers in the network improved the classification accuracy. In this implementation we added two additional 3D convolution layer between the existing Conv and FC layers. The architecture is detailed in Fig. 3.

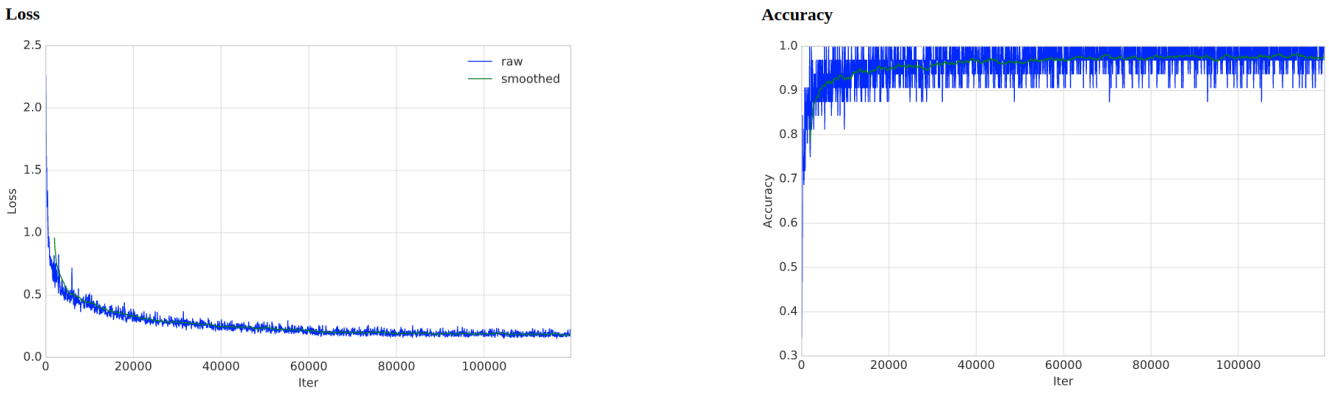
C. SVM Model Selection

The importance of model selection and hyperparameter setting is well-known for machine learning models. To select the best available SVM model and hyperparameters for this task and at the same time avoid over-fitting, we did an extensive grid search cross validation for kernels, i.e., linear and rbf (radial basis function), hyper-parameters C and γ (hyperparameter of rbf) of the model on validation data. The searching space of each hyperparameter is updated each time by re-spreading it around the optimal value obtained in the last grid search. The final round of grid search cross validation results are given in Table III where the best hyper parameters are highlighted in bold.

Avg Acc (Var)	**kernel**	C	γ
0.614 (+/-0.016)	rbf	100	1.0
0.870 (+/-0.008)	**rbf**	**100**	**0.1**
0.834 (+/-0.017)	rbf	100	0.01
0.614 (+/-0.016)	rbf	1000	1.0
0.870 (+/-0.010)	rbf	1000	0.01
0.828 (+/-0.018)	rbf	1000	0.01
0.804 (+/-0.016)	**linear**	**1.0**	-
0.802 (+/-0.011)	linear	0.1	-
0.735 (+/-0.009)	linear	0.01	-

TABLE III: Grid Search Cross Validation on ModelNet10 Validation Data.

Theoretically, the grid search cross evaluation should be conducted on training data instead of validation data. However, in practice, the high-dimensional and large-scale data makes k-fold cross evaluation computational expensive. Instead, we conducted it only on validation data which is assumed to have the same distribution as training data. This assumption also makes sense as the validation data is uniformly and randomly split from



(a) Training loss

(b) Training accuracy

Fig. 4: Training curves of the Theano baseline

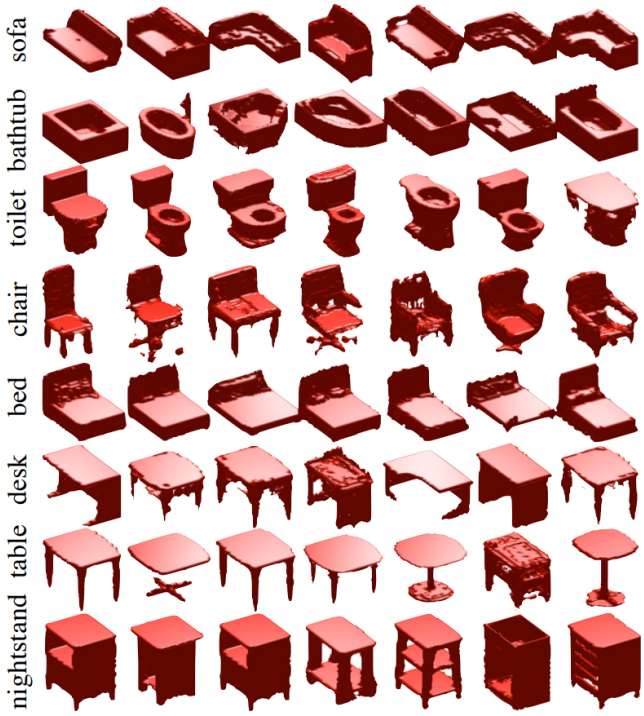


Fig. 5: Few examples from ModelNet10 dataset [20]

the training data. After cross validation, we assume the best hyper-parameters are also valid for training data.

As we can see from Table III, models with radial basis function kernel generally have better results with suitable hyperparameters compared with linear models. This result is aligned with our expectation as the squared exponential kernel in radial basis function defines a function space that is a lot larger than that of the linear kernel. It actually intrinsically maps the input data to a infinite-dimensional space where it is intuitively easier to separate data of one class from other classes.

D. Multi-classification Strategies Comparison

After hyperparameters are tuned by grid search cross validation, we compare the performance of different multi-classification strategies, i.e., one-vs.-one (ovo), one-vs.-rest (ovr) and crammer singer [6], on a linear SVM model with $C = 0.3$. The comparison results are given in Table IV.

Approach	Avg Acc (Var)
ovo	**0.818**
ovr	0.801
crammer˙singer	0.809

TABLE IV: Comparison of performance under different multi-classification strategies.

From the results, we can draw the conclusion that one-vs.-one strategy works the best among those three strategies for our model. But we are also aware that this improvement on performance is at the cost of higher computational complexity, as one-vs.-one strategy builds 45 binary classifiers for 10 object classes whereas one-vs.-rest approach only need to build 10 classifiers. Therefore, there is a trade-off between classification accuracy and the amount of computation.

E. SVM Evaluation on ModelNet10

We have obtained the best hyperparameters and multi-classification strategy of our SVM model from cross validation on validation data. Then we make these choices and re-train the best model using all the training data to learn the optimal weights of this model. To evaluate the performance of SVM on object recognition, we train and test it on ModelNet10 dataset, and report the performance in different metrics.

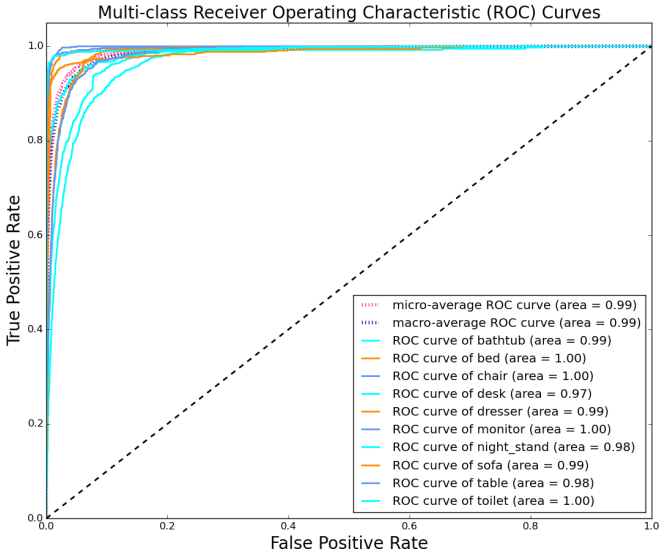
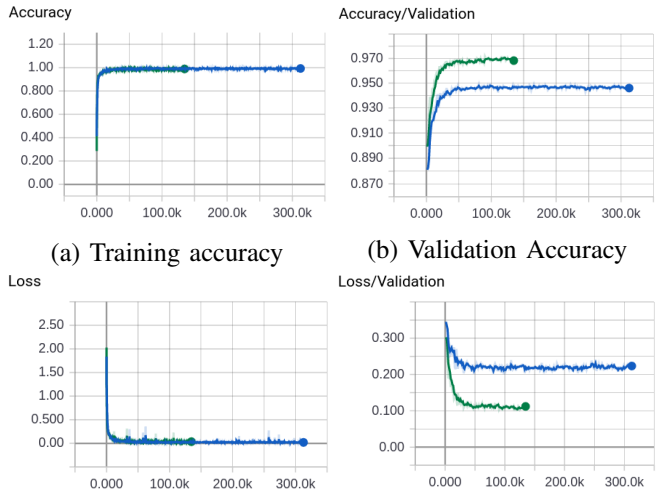


Fig. 6: ROC curves of SVM Model on ModelNet10



(a) Training accuracy (b) Validation Accuracy

(c) Training Loss (d) Validation Loss

Fig. 7: Training curves from tensorboard at every training step. **Blue**- Voxnet(baseline) **Green**- Our network

A *confusion matrix* is a table that describes the performance of a classifier on a set of test data in a straight-forward manner. It illustrates the relation between true class labels and the predicted labels from a classifier. The diagonal elements of this matrix are truth positives (TP) and truth negatives (TN) whereas other elements are false positives (FP) and false negatives (FN). The confusion matrix of our SVM model on ModelNet10 is given in Fig. 8c.

We also computed probabilities of possible outcomes for test data, as soft-labels are preferred than hard-labels in many applications especially in the field of robotics. Receiver Operating Characteristic (ROC) curves are metrics to evaluate classifier with probabilistic outputs by computing TP rate and FP rate under varied discrimination thresholds instead of a fixed threshold. The ROC curves of SVM model for 10 classes on ModelNet10 test data are plotted in Fig. 6. The larger the Area Under an ROC Curve (AUC), the better performance it indicates for that object class.

F. Performance Comparison of Three Models

We finally compare the performance of SVM model with two variations of 3D CNN model we implemented on average accuracy in Table V. As we can see from the table, SVM model outperforms 3D CNN model by 0.44%. The deep 3D CNN model has the best object recognition performance among these three models with average accuracy of 89.72%. The conclusion of this experiment is that replacing the fully-connected layer with a soft kernel SVM with tuned hyperparameters only marginally improved the performance of the original 3D CNN model, however, increasing the number of layers to make it deeper indeed made a larger improvement on the object recognition accuracy.

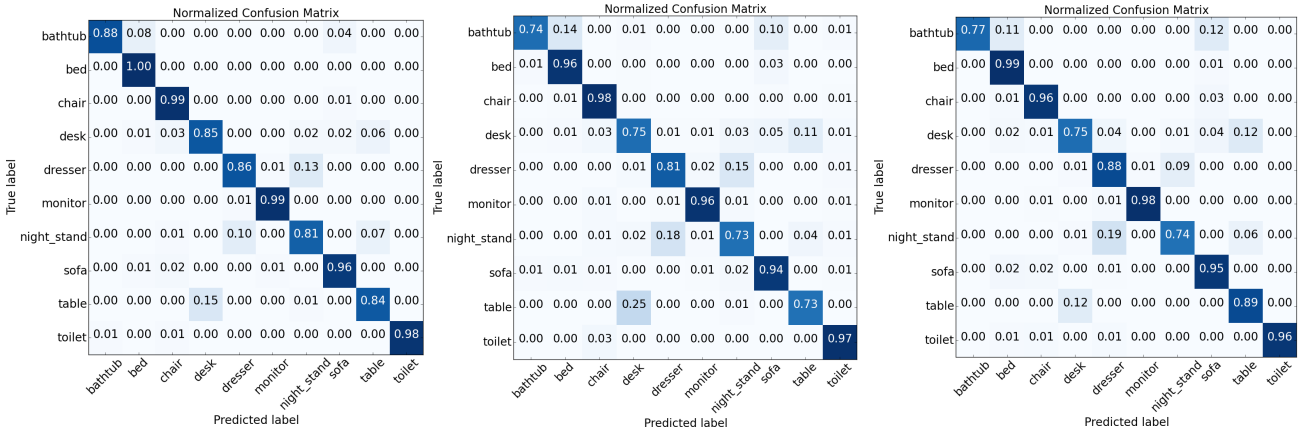
Method	**Avg Acc**
SVM Model	0.8686
3D CNN Model	0.863
Deep 3D CNN Model	**0.8972**

TABLE V: Performance comparison of three models on ModelNet10 test data.

V. Conclusions

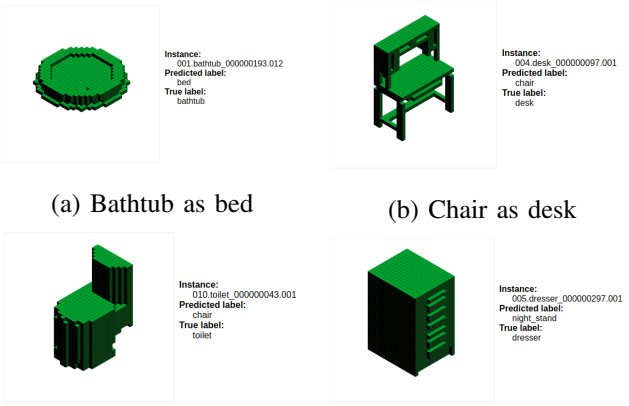
We successfully implemented the baseline in tensorflow from scratch and found similar performance as reported. As part of the ambitious goals, we also implemented two additional methods in order to improve performance. As discussed in the previous section, we found a marginal classification improvement with additional SVM layer. However, in the case of additional convolution layer, we found a significant improvement in the performance ($\sim 3.5\%$). This indicates that deeper network perform better than their shallower counterpart.

We also found some misclassified labels as shown in Fig. 9. We concluded those misclassified images are ambiguous even for humans to distinguish. However, if we do have time to train the network even better or make the network deeper, we believe the number of misclassified labels would be lesser.



(a) Theano (baseline) (b) SVM using NN features (ours). (c) Deep NN (ours) .

Fig. 8: Confusion Matrix of Three Models on ModelNet10



(a) Bathtub as bed (b) Chair as desk

(c) Chair as toilet (d) Nightstand as dresser

Fig. 9: Some examples of misclassified objects

VI. Implementation Details

Our neural network architectures were implemented in TFLearn [7] with Tensorflow [1] backend. The models were trained and tested on Ubuntu 16.04.2 LTS system with Intel Core i7-5820K CPU and Nvidia GeForce GTX 1080 8GB GPU and took between 1.5-2 hours to train.

We have made the source code available online `https://github.com/abhven/ML-Project`

VII. Team Contribution

At the starting of the project, we had broken down the project into smaller tasks and assigned each of the tasks to the team members. Though we had a little reshuffling from the initial proposal, in the end each of our contribution was roughly equal. Below are the exact contribution by each of the team members.

- Execute author's code (Theano based) and compare the results mentioned in the paper - *Bruce*.
- Preprocessing data from training/testing- *Abhishek*
- Implementating base-line (exactly as paper) from scratch in tensorflow- *Mrunmayee*
- Ambitious goal 1: Replace final layer with SVM and evaluate performances- *Lu*
- Ambitious goal 2: Use a deeper neural network and evaluate performances- *Abhishek and Lu*
- Evaluation and discussion of results; Report- *All Members*

REFERENCES

[1] J. Allaire, D. Eddelbuettel, N. Golding, and Y. Tang, *tensorflow: R Interface to TensorFlow*, 2016. [Online]. Available: https://github.com/rstudio/tensorflow

[2] S. Bai, X. Bai, Z. Zhou, Z. Zhang, and L. Jan Latecki, "Gift: A real-time and scalable 3d shape search engine," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5023–5032.

[3] C. Bishop, "Pattern recognition and machine learning (information science and statistics), 1st edn. 2006. corr. 2nd printing edn," *Springer, New York*, 2007.

[4] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *arXiv preprint arXiv:1608.04236*, 2016.

[5] S. Choudhury, S. Arora, and S. Scherer, "The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety," in *AHS 70th Annual Forum*, 2014.

[6] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of machine learning research*, vol. 2, no. Dec, pp. 265–292, 2001.

[7] A. Damien *et al.*, "Tflearn," https://github.com/tflearn/tflearn, 2016.

[8] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results," http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[10] V. Hegde and R. Zadeh, "Fusionnet: 3d object classification using multiple data representations," *arXiv preprint arXiv:1607.05695*, 2016.

[11] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," in *Robotics Research*. Springer, 2017, pp. 235–252.

[12] E. Johns, S. Leutenegger, and A. J. Davison, "Pairwise decomposition of image sequences for active multi-view recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3813–3822.

[13] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[14] D. Maturana and S. Scherer, "Voxnet: A 3d convolutional neural network for real-time object recognition," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 922–928.

[15] M. Pontil and A. Verri, "Support vector machines for 3d object recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 6, pp. 637–646, Jun 1998.

[16] B. Shi, S. Bai, Z. Zhou, and X. Bai, "Deeppano: Deep panoramic representation for 3-d shape recognition," *IEEE Signal Processing Letters*, vol. 22, no. 12, pp. 2339–2343, 2015.

[17] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, "Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[18] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[19] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling," in *Advances in Neural Information Processing Systems*, 2016, pp. 82–90.

[20] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, J. Xiao, and X. Tang, "3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.