3D Object Recognition with Deep Networks

Adrian Schneuwly
adrischn@ethz.ch

Johannes Oswald
voswaldj@eth.ch

Tobias Grundmann
tobiagru@ethz.ch

Abstract

We implement 3D object recognition using a Convolutional Neural Network (CNN) approach. We adopt the network structure outlined in [5] and implement it in Python using the Keras framework. We compare our trained CNN with the original paper and achieve similar positive results.

1. Introduction

Autonomous robotic systems like self-driving cars or drones operating in real world environments strongly depend on robust object recognition in order to get a semantic understanding of the immediate surroundings. Meanwhile, inexpensive depth data has become available with active range sensor such as LiDAR and RGBG cameras (e.g Google Tango). This information can aid in a wide range of tasks. While it is commonly used for obstacle avoidance and mapping, it is not fully utilized in the task of object recognition.

Machine learning approaches (Fig. 1) in the form of Convolutional Neural Networks (CNN) dramatically improved state of the art image recognition [4]. Such networks learn the feature and classifiers from given training data. While it is simple to extend this basic approach to volumetric data, it is not obvious what network structures will yield good performance. VoxNet [5] is a fast basic 3D CNN architecture that makes use of the available depth information and achieves high object detection accuracy for 3D point clouds.

2. Related Work

On the one hand, there is a large body of work of object recognition using 3D point clouds but not making use of Neural Networks. Mostly methods combine hand-crafted features or descriptors with a machine learning classifier ([10], [11], [12]). Similiar methods are seen with semantic segmentation, with structured output classifiers instead of single output classifiers ([14], [15], [16]).

On the other hand 2.5D CNNs were used for object recognition but fail to make full use of geometric information in the data. Their approaches simply treat the depth channel as an additional channel, along with the RGB channels. ([17], [18], [19], [20])

Figure 1. Example of object recognitions: Detection as binary classification problem (pedestrian vs no pedestrian). Slide over all possible locations in the image and classify patches. (Image source: [3])

Additionally, 3D CNNs already proved there usability in video analysis ([23], [24]). In this case, time acts as the third dimension but algorithmically, these architectures work the same as ours, but the nature of the data is very different.

TODO: Das so machen? ist quasi kopiert von VoxNet. Finde ich ok, mir egal

3. Convolutional Neural Networks

At the core of a neural network is a logistic classifier. For given data, we train weights and bias of a linear equation which will then be transformed into probabilites by using a softmax function. Figure 3.

Linear models are limited and therefore we deepen the network in our case through convolutions and max pooling. For convolutions, we run small neural networks with given output size/ depth on a patch which walks through our 3D Data. When leaving out /jumping voxels (striding), the spatial dimension is squeezed while extracting local features of the given object. Through repeating this procedure we arrive at an output depth which coresponds to the semantic complexity of our object recognition task.

The second technique to deepen our network is pooling. Instead of striding to sqeeze our dimension, we can extract

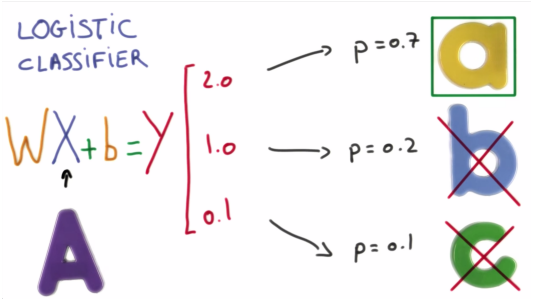

Figure 2. Weight and bias (W,b) get trained to classify input A. Source: [3]

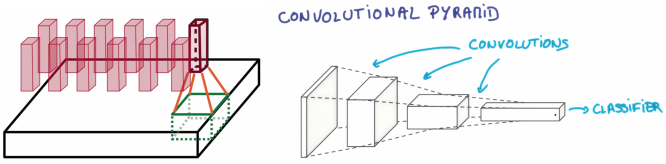

Figure 3. Left: Patch with stride 2 Right: Multiple Covolutions. Source: [3]

e.g. the maximum of a patch and therefore reduce spatial dimension. See detailed discribition of model used in our approach in Figure 5.

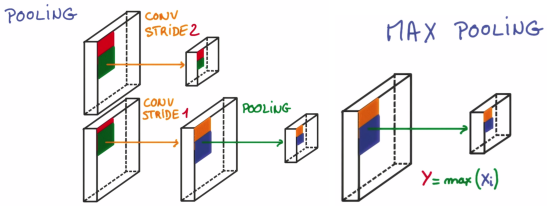

Figure 4. Left: Stride 2 vs. Stride 1 + Pooling Right: Max-Pooling. Source: [3]

In order to train our model, we have to test how well it is doing during and after training. Validating the model on training data does not work since the model remembers all its input. Therefore we split the data into test- and training set which enables us to check how accurate the classifier is doing on unknown input, the test set. This common phenomenon, that a complex model fits a given dataset but fails to generalizes on new data, is called *overfitting*.

TODO: Overftting hier drin, weil das der Grund ist fr die nicht so guten Ergebnisse von uns? Irgendwas zu batch size, epochen, mehr details erzhlen

4. Data: Princeton ModelNet

The data on which we train our Deep Network was build by [6]. After a list of the most common object categories in the world was compiled from Princtons SUN database [2], 3D CAD models were collected through online search engines. To verify the object assigment to each category, human workers on Amazon Mechanical Turk were hired to validate manually. We trained our neural network on one smaller and one larger dataset containing 10 respectivally 40 categories which each ?XXXX? objects.

5. Reimplementing: VoxNet [5]

In the following, parts contributed by Author 1/2/3 will be labeled as A1/A2/A3.

Overall Goal is to classify an object from a given point cloud. After voxilizing to format 32x32x32, we train our CNN on these occupancy grids, Figure 5.

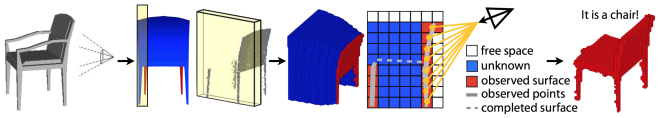

Figure 5.
Object - Depth & Point Cloud - Occupancy Grid - Classification. Source: [6]

Princetons ModelNet delivers every object with views from 12 different angles in .mat files; we train on all rotations to enhance recognition from different views. The data is already voxilized to 32x32x32 occupancy grids. This data was transformed by A3 into a .hdf5 format and split into test and training set. Hdf5 compresses the set size dramatically and enables simple access. Two methods were implemented by A1/A3 to deal with loading the data from .hdf5 files. One opens the .hdf5 and loads pieces which are passed during iterations of the generator resulting in a slower but RAM saving loader. The other, used for the final training, converts the hdf5 entirely to a numpy array providing us with a RAM expensive but much faster access to the data.

After preparing the data and its access, the VoxNet convolutional neural network model (900.000 Parameters) is created, Figure 5. For this, A1/A2 used the *Python* deep learning library *Keras* which runs on top of *Theano*. *Keras* supports 3D convolutional and max-pooling layers.

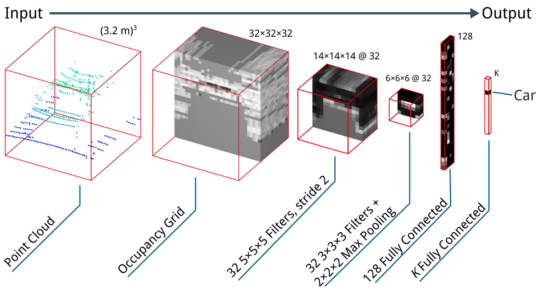

Figure 6. VoxNet Convolution Model. Source: [1]

A training enviroment was set up and used by A1/A2/A3.

6. Experiments

The training process takes around 9 to 20 hours on a NVIDIA GTX 980TI (6GB) GPU depending on the choice of the dataset (ModelNet10/40).

TODO: Mehr...

7. Results

The papers implementation achieved verg good results for the small data set but failed to achieve the excelent score of the original VoxNet implementation on the larger ModelNet40. This is due ... **TODO: Mehr & Warum?**

Algorithm	ModelNet10 Classification Accuracy	ModelNet40 Classification Accuracy
VoxNet [5]	83%	92%
3DShapeNets [6]	77 %	83.5%
ETH VoxNet	**81.8%**	**82.3 %**

8. Conclusion

The team successfully reimplemented a very clean and working VoxNet with comparable results. **TODO: Mehr...**

References

[1] D. maturana website. `http://www.dimatura.net/pages/3dcnn.html`.

[2] Sun database. `http://sun.cs.princeton.edu/`.

[3] Udacity deep learning. `https://www.udacity.com/course/deep-learning--ud730`.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] D. Maturana and S. Scherer. VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In *IROS*, 2015.

[6] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *CVPR*, 2015.