Convolutional Neural Networks for 3D MNIST Image Classification

Jeremy Irvin
Stanford University
jirvin16@stanford.edu
https://github.com/jirvin16/3D-MNIST-Classification

Abstract

Visual object recognition is an essential skill for autonomous robots to function in real world environments. As LiDAR and RGBD cameras are more commonly used in autonomous vehicles [1,2,3], automatic, real-time processing of 3D data in these vehicles is becoming more and more necessary. Classification of 3D objects is a step in this direction. Moreover, this classification must be robust to noisy images from varying perspectives. In this work, we present a model which efficiently and robustly classifies a 3D version of the MNIST dataset with random noise and rotations as augmentations [4]. We present several experimental results, of which the best 3D Convolutional Neural Network architecture achieves 72.8% accuracy on the test set with, beating all baselines by a significant margin.

1 Introduction

The availability of 3D data has been increasing rapidly as the use of sensors which provide 3D spatial readings in autonomous vehicles and robots becomes more prevalent. For this reason, the ability of these machines to automatically identify and make inferences about this data is a crucial skill for them to operate autonomously. We focus on classification in a niche setting as a demonstration of a success in this direction. Furthermore, this classification must be robust to perspective variations as well as noise, as data output by these sensors is often noisy.

Convolutional Neural Networks have demonstrated immense success recently in the field of computer vision [5]. The MNIST dataset in particular has been effectively classified using architectures of this type, with current state-of-the-art at a high 99.79% accuracy [6]. The invariances of the convolutional architectures to translation and rotation allow it to outperform all other models in this task and many other computer vision tasks as well. This serves as inspiration for generalizing these 2D methods into 3D [7]. This work does not propose this generalization, but applies it to a recently constructed 3D dataset.

A new dataset of 3D point clouds has been created from the original 2D MNIST dataset [4]. In this work we apply 3D Convolutional Neural Networks in order to perform classification of this 3D MNIST dataset. The structure of our network is very similar to VoxNet, which demonstrated great success in 3D real-time object recognition on different datasets [8].

2 Related Work

3D object classification is relatively well-studied area, with data typically coming from RGBD and LiDAR sensors. Many of the successful methods use hand-crafted features with a machine learning classifier [9, 10, 11, 12, 13]. However these features are often inefficient to compute or are specific to the type of data being computed by the sensor. Features computed on point clouds can rely on nearest neighbor searches which quickly become intractable as the number of points becomes large. To

combat this, several methods have been proposed which use deep learning to automatically extract features from general 3D point clouds and their volumetric representations [16, 17]. 3D ShapeNets use a Convolutional Deep Belief Network with weight sharing and no pooling layers to perform 3D object classification on CAD models [16]. Later a new more accurate and more efficient deep learning model known as VoxNet was proposed to convert point clouds to volumetric representations and then apply 3D convolutions [17]. In this work we use a very similar data pipeline and model architectures to VoxNet. It is important to note that 3D Convolutional Neural Networks were first proposed for video tasks, where the third dimension is temporal [7]. These models do not rely on this fact and can be applied to data where the third dimension is spatial, as in this work. They are a simple generalization of the 2D convolution as explained below in Section 4.

3 Data

A portion of the original 2D MNIST dataset (around 5000 images) has been converted to a dataset of 3D point clouds, ie, sets of (x, y, z) coordinates. In order to encourage the classifier to learn more robust features, the data has been augmented by adding randomly rotated and noisy data generated from the original 3D images [4] More specifically, each image in the original dataset of 5000 3D point clouds was used to create another image with a rotation in the x- and y-axis selected from a normal distribution with mean 0 and variance 90 and z-axis selected from a normal distribution with mean 0 and variance 180. The total dataset after this augmentation consists of 10000 images. Examples of these perturbations along a single axis can be seen in Figure 1.

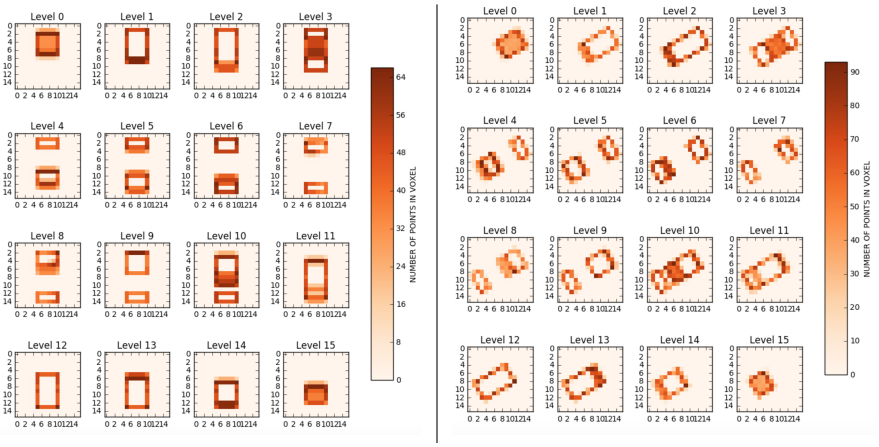

Figure 1: Example of rotation clockwise by $60°$ along the z-axis. The left panel shows a projection of an image of the digit 5 into the x dimension (the y-axis of every image is the y-dimension, the x-axis of every image is the z dimension, and the levels are the x dimension. The right panel shows the image rotated clockwise by $60°$ in the z dimension. The colors denote the number of points within a voxel.

Rather than operating on point clouds directly, we first transform each image into their volumetric representation using a transformation known as *voxelization* [14]. This is the process of taking a 3D cloud and fitting an axis aligned rectangular prism (box) around it (using the minimum and maximum values in every dimension of the image), subdividing the box in segments, and finally assigning each point in the point cloud one of the subboxes (known as voxels, analogous to pixels). Figure 2 provides an illustration of this process, and a visualization of a voxelized 3D point cloud can be seen in Figure 3. . This transformation maps all 3D images into the same fixed-dimensional representation. We subdivide each dimension into 16 segments, resulting in a $16 \times 16 \times 16$ dimensional array for each image, where each entry denotes the number of points in the voxel corresponding to that entry[1]. Note that the points in the point cloud do not have RGB values, so we can think of this image as a 3D image with a single channel (as opposed to three channels of a usual RGB image). The baselines use a 'flattened' version of the representation, meaning the $16 \times 16 \times 16$ array flattened into a $16 * 16 * 16 = 4096$ dimensional vector.

[1]We tested splits into 8 and 32 segments as well, but neither yielded comparable performance.

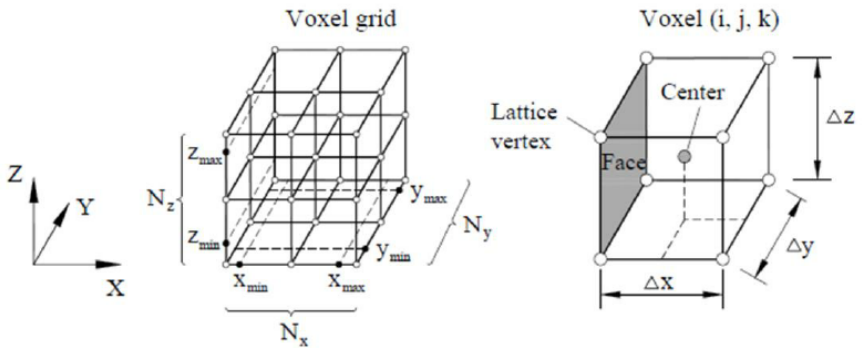

Figure 2: Voxelization. Transforming a point cloud into a fixed-dimensional representation.

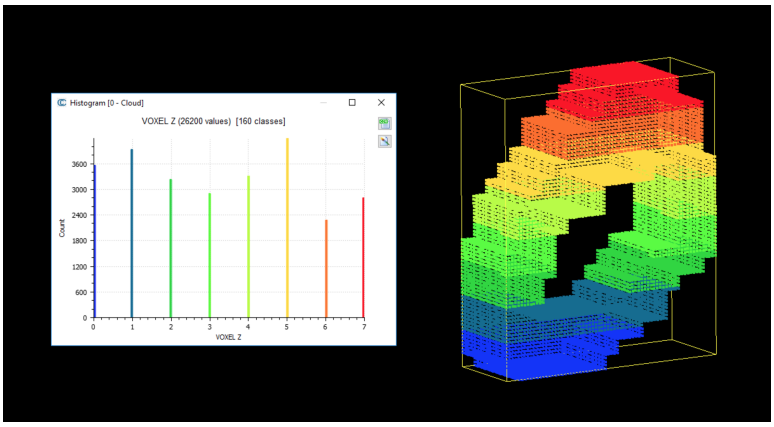

Figure 3: The right image is an example of the digit 0 split into 8 voxels along the z-axis (each color corresponds to a single voxel). The left graph is a count of points in each voxel in the z-dimension (the colors match the corresponding colored voxel in the 3D visualization).

4 Model

All of the baselines depend heavily on the 'flattening' data transformation, which removes the notion of spatial locality from the dataset. To combat this, we use different 3D convolutional neural network architectures with 3D max pooling. For these 3D convolutional networks to be effective, we must use data which retains the original spatial locality, namely the raw data before 'flattening' - the voxelized 3D point cloud.

The main components of the architectures include 3D convolutional layers as well as 3D max pooling layers. The 3D convolutional layers consist of cubic (3-dimensional) filters to pass over the 3-dimensional matrix with shared weights across the image. This is simply a 3-dimensional windowed-rolling dot product over the image. The 3D max pooling layers are applied after the convolution operation. This is just a 3-dimensional windowed-rolling maximum over the image. As in the 2D setting, these operations are designed to capture invariances in the image. This can be seen by noticing that in the filtering operation, the weights are shared, so each filter will commonly 'activate' by a certain set of input values (features). Pooling 'blurs' the feature extractors by only considering the maximum values in a region. Using both of these operations as consecutive layers will ideally capture translational and scaling invariances in the images. Figure 4 illustrates the general filter and pooling operation, and Figure 5 shows an example of a single layered 3D convolutional architecture.

The overall architectures consist of stacks of convolutions (filters), leaky rectified linear units (ReLu), and max pools, and then a fully connected layer on the 'flattened' output of the last layer of the convolutional network into a hidden dimension size, into a sigmoid nonlinearity and a final fully

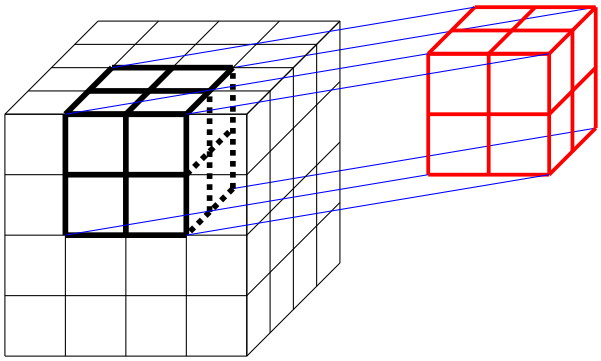

Figure 4: 3D Filter and Pooling operations on a simple $4 \times 4 \times 4$ image. The $2 \times 2 \times 2$ cube on the right can be thought of as either the filter or the pool. This figure does not show the stride parameter - this is just the size of the shift of the filter or pool (a sliding window). The blue lines indicate a dot product over all 3 dimensions in the case of a filter, or a max operation in the case of max pooling. The result is a scalar in every instance of the operation.

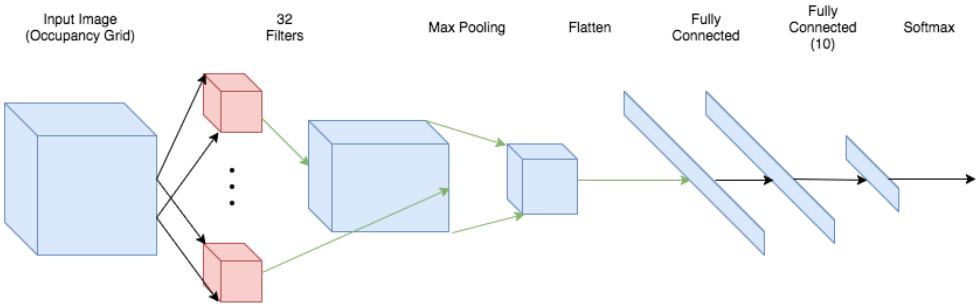

Figure 5: Example of basic network architecture with a single convolutional layer. One convolutional layer, max pooling layer, fully connected layer, and last fully connected layer into the 10 classes, where a final softmax is performed. Cross entropy loss is used and trained in Tensorflow with an Adam optimizer.

connected layer into the number of classes (10) followed by a softmax. The following describes a generic two layer network (as a function of a single training example $x^{(i)}$)

$$\begin{aligned} h_1(x^{(i)}) &= g\left(W_2 \circledast (W_1 \circledast x^{(i)})\right) \\ h_2(x^{(i)}) &= W_4 \cdot \sigma\left(W_3 \cdot h_1(x^{(i)}) + b_3\right) + b_4 \\ f_{y^{(i)}}(x^{(i)}) &= \text{softmax}\left(h_2(x^{(i)})\right) \end{aligned}$$

where W_1, W_2 are the filter weights, $\circledast$ denotes the 3D convolution operation, g is the 3D max pooling operation, W_3, W_4 are weight matrices, b_3 and b_4 are biases, $\cdot$ denotes regular matrix multiplication, σ denotes the sigmoid nonlinearity (which is swapped with different activation functions in the experiments), and softmax denotes the softmax operation. The loss optimized is cross entropy, namely (again for a single training example $(x^{(i)}, y^{(i)})$),

$$L(x^{(i)}, y^{(i)}) = -\log\left(\frac{\exp\left(f_{y^{(i)}}(x^{(i)})\right)}{\sum_j \exp(f_j(x^{(i)}))}\right)$$

where $f(x^{(i)})$ is the output of the softmax layer and $f_{y^{(i)}}(x^{(i)})$ is the $y^{(i)}$th index of this output. This is the cross-entropy $H(p^{(i)}, q^{(i)})$ between the estimated class probability distribution $p^{(i)}$ output by the softmax operation and the true distribution $q^{(i)} = [0 \cdots 1 \cdots 0]$ with a 1 in the $y^{(i)}$th position. This loss is standard for multiclass classification in neural networks.

5 Experiments

We have implemented several baselines, including variants of multiclass SVM, multinomial logistic (aka softmax) regression, and neural networks with a softmax layer as the final output layer. Figure 6 outlines more explicit model evaluations. The oracle is a vanilla CNN which cheats and uses the original 2D image (from MNIST) that was used to create the 3D point clouds.

Model	Test Accuracy
Linear multiclass ovr SVM: L2 Regularization, Squared Hinge Loss	0.554
Linear multiclass ovr SVM: L1 Regularization, Squared Hinge Loss	**0.566**
RBF kernel multiclass ovr SVM	0.542
Polynomial kernel multiclass ovr SVM	0.126
Sigmoid kernel multiclass ovr SVM	0.51
ovr Logistic Regression:	**0.5905**
Multinomial Logistic Regression	0.583
2 Layer Neural Network, 128 Hidden Dimension, Sigmoid Nonlinearity	0.622
2 Layer Neural Network, 256 Hidden Dimension, Sigmoid Nonlinearity	**0.634**
2 Layer Neural Network, 512 Hidden Dimension, Sigmoid Nonlinearity	0.6315
2 Layer Neural Network, 1024 Hidden Dimension, Sigmoid Nonlinearity	0.6285
Oracle (Vanilla CNN)	0.992

Figure 6: Summary of baseline results on test set (2000 examples). ovr stands for *one-versus-rest* in contrast with a *one-versus-one* scheme. The best SVM, logistic regression, and neural network classifiers are bolded. The oracle is a vanilla convolutional neural network which uses the 2D image that was used to generate the 3D point clouds.

Many different architectures were tested, including varying number of layers, number of filters, filter sizes, fully connected hidden dimension size, and activation functions [2]. All of the weight matrices were initialized from a normal distribution with standard deviation 0.01, and all of the biases were initialized to a constant 0.1 elementwise. The leaky ReLU units after each convolutional layer used $\alpha = 0.1$. A dropout of 0.2 was used between the convolutional layers (in the case of a 2-layer network), after the max pooling layer, and between the fully connected layers. The entire network is optimized end-to-end using Adam [15] with a fixed learning rate of 0.001 with TensorFlow. Moreover, a model with 2 layers, 32 filters, and a fully connected hidden dimension of 256 was tested with several different hidden nonlinearities between the fully connected layers. Each model takes less than 30min to train on a GPU and the network can classify a single example in milliseconds.

6 Results

A summary of the experimental results (using a sigmoid as the nonlinearity between the fully connected layers) on the test set can be seen in Figure 7. Models with 32 filters marginally outperformed the models with 16 filters. The best model has 2 layers, 32 filters, and a 512 hidden dimension in the fully connected layers. Figure 6 shows a comparison of nonlinearities between the fully connected layers. Sigmoid performed the best and was used for all of the other experiments. Moreover, each of the models beat the baselines by a significant margin. None, however, were able to compete with the oracle.

The success of the 3D convolutional neural networks on this dataset verifies the intuition stated before - the model is robust to rotation and noise. The best architecture is able to beat the best baseline model by an accuracy of 10%, a significant margin. The best mode's rotational invariance is visualized in Figure 9. Most of the activations are consistent across all of the angles of rotation, demonstrating that the model is robust to rotation across different axes. The baselines, however, do not share this robustness. The baselines are likely unable to model invariance to rotations likely due to the flattening operation which is required as a preprocessing step to input into these models.

[2] 64 filters were tested as well but performed much poorer than 16 or 32.

Moreover, the convolutional neural network architectures train and run very efficiently. The confusion matrix of the best model's mistakes on the test set can be seen in Figure 10. Notice that the majority of the mistakes were sensible. For example, the model often confuses 2 and 7, 3 and 8, 6 and 9, and 3 and 5.

Model (Sigmoid)	Test Accuracy
1 layer, 16 filters, 128 hidden dim	0.6535
1 layer, 16 filters, 256 hidden dim	0.66
1 layer, 16 filters, 512 hidden dim	0.675
2 layer, 16 filters, 128 hidden dim	0.7
2 layer, 16 filters, 256 hidden dim	0.696
2 layer, 16 filters, 512 hidden dim	0.715
2 layer, 16 filters, 1024 hidden dim	0.726
1 layer, 32 filters, 128 hidden dim	0.6655
1 layer, 32 filters, 256 hidden dim	0.6915
1 layer, 32 filters, 512 hidden dim	0.6475
2 layer, 32 filters, 128 hidden dim	0.7125
2 layer, 32 filters, 256 hidden dim	0.7275
2 layer, 32 filters, 512 hidden dim	**0.728**
2 layer, 32 filters, 1024 hidden dim	0.7225

Figure 7: Summary of results of different 3D Convolutional Neural Network architectures.

Model (2 layer, 32 filters, 256 hidden dim)	Test Accuracy
ReLU	0.7055
Sigmoid	**0.7275**
Tanh	0.719
ELU [5]	0.7105
None	0.689

Figure 8: Comparison of nonlinearities between fully connected layers.

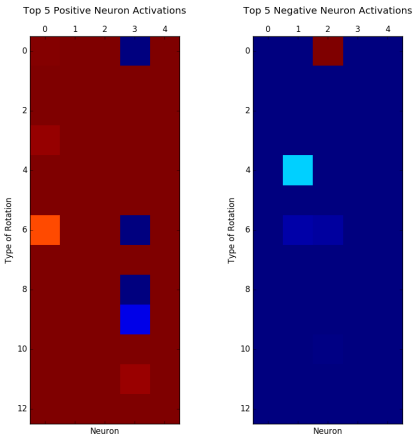

Figure 9: The activations of the top 5 most active (left) and inactive (right) neurons on the rotated examples in the test set when passing in an image of the digit 0 with 12 different rotations, namely $-60, -30, 30, 60$ degree rotations across each axis (rows 1-4 are x-axis rotations, rows 5-8 are y-axis rotations, and rows 9-12 are z-axis rotations). Red means high activation, blue means low activation. The first row in each panel are the activations of the original image

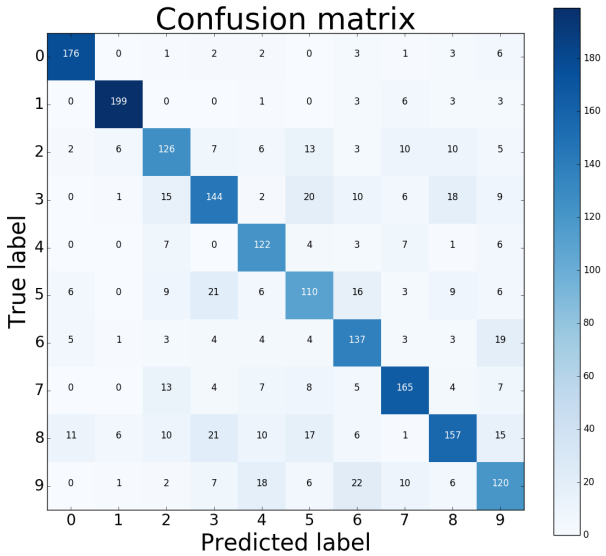

Figure 10: Confusion matrix of the best model on the test set.

7 Conclusion

The 3D convolutional neural network architectures can successfully extract features from the volumetric representations of the 3D MNIST dataset. They outperform the baselines in almost every experiment, achieving a high 72.8% accuracy. Moreover these models are robust to rotation and noise as demonstrated by the success on this perturbed dataset as well as the visualization of the activations in Figure 9. However, there is still a large gap between the best model proposed here and the oracle. Future work will attempt to close this gap, by running models on larger datasets (the original MNIST dataset was subsampled when generating the 3D dataset). Additionally, different occupancy grid transformations should be tested, such as probabilistic ones as used in [17].

8 References

[1] C. Urmson and J. Anhalt abd H. Bae abd J. A. D. Bagnell and C. R. Baker and R. E. Bittner and T. Brown and M. N. Clark and M. Darms and D. Demitrish, J. M. Dolan and D. Duggins and D. Ferguson and T. Galatali and C. M. Geyer and M. Gittleman and S. Harbaugh and M. Hebert and T. Howard and S. Kolski and M. Likhachev and B. Litkouhi and A. Kelly and M. McNaughton and N. Miller and J. Nickolaou and K. Peterson and B. Pilnick and R. Rajkumar and P. Rybski and V. Sadekar and B. Salesky and Y.-W. Seo and S. Singh and J. M. Snider and J. C. Struble and A. T. Stentz and M. Taylor and W. R. L. Whittaker and Z. Wolkowicki and W. Zhang and J. Ziglar. Autonomous driving in urban environments: Boss and the urban challenge. *JFR*, vol. 25, no. 8, pp. 425–466, Jun. 2008.

[2] A. S. Huang and A. Bachrach and P. Henry and M. Krainin and D. Maturana and D. Fox and N. Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. *ISRR*, Flagstaff, Arizona, USA, Aug. 2011.

[3] S. Choudhury and S. Arora and S. Scherer. The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. *AHS*, May 2014.

[4] kaggle.com/daavoo/3d-mnist

[5] A. Krizhevsky and I. Sutskever and G. Hinton. ImageNet classification with deep convolutional neural networks. *NIPS*, 2012.

[6] L. Wan and M. Zeiler and S. Zhang and Y. L. Cun and R. Fergus. Regularization of neural networks using dropconnect. *ICML*, 2013.

[7] S. Ji and W. Xu and M. Yang and K. Yu. 3D convolutional neural networks for human action recognition. *ICML* , 2010.

[8] http://www.dimatura.net/extra/voxnet_maturana_scherer_iros15.pdf

[9] T. Du and P. Xu and R. Hu. CS 221 Final Report: 3D Shape Classification. 2015.

[10] A. Frome and D. Huber and R. Kolluri. Recognizing objects in range data using regional point descriptors. *ECCV*, vol. 1, 2004.

[11] J. Behley and V. Steinhage and A. B. Cremers. Performance of histogram descriptors for the classification of 3D laser range data in urban environments. *ICRA*, 2012.

[12] A. Teichman and J. Levinson and S. Thrun. Towards 3D object recognition via classification of arbitrary object tracks. *ICRA*, 2011.

[13] A. Golovinskiy and V. G. Kim and T. Funkhouser. Shape-based recognition of 3D point clouds in urban environments. *ICCV*, 2009.

[14] https://en.wikipedia.org/wiki/Voxel

[15] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. 2014.

[16] Z. Wu, S. Song and A. Khosla and X. Tang and J. Xiao. 3D ShapeNets: A deep representation for volumetric shape modeling. *CVPR*, 2015.

[17] D. Maturana and S. Scherer. VoxNet: A 3D convolutional neural network for real-time object recognition. *IROS*, 2015.