Sparse 3D convolutional neural networks

Ben Graham
b.graham@warwick.ac.uk

Department of Statistics
University of Warwick
CV4 7AL, UK

Convolutional neural networks (CNNs) are powerful tools for understanding data with spatial structure such as photos. They are most commonly used in two dimensions, but they can also be applied more generally. One-dimensional CNNs are used for processing time-series such as human speech. Three dimensional CNNs have been used to analyze movement in 2+1 dimensional space-time [2] and for helping drones find a safe place to land [3]. Three dimensional convolutional deep belief networks have been used to recognize objects in 2.5D depth maps [4].

In [1], a *sparse* two-dimensional CNN is implemented to perform Chinese handwriting recognition. When a handwritten character is rendered at moderately high resolution on a two dimensional grid, it looks like a sparse matrix. If we only calculate the hidden units of the CNN that can actually *see* some part of the input field the pen has visited, the workload decreases.

Sparsity is a useful optimization in two dimensions, and it is potentially even more useful in three or higher dimensions. This is related to the *curse of dimensionality*; an $N \times N \times N$ cubic grid contains many more points than an $N \times N$ square grid. We have adapted the algorithm from [1] to implement sparse CNNs on range of different graphs. CUDA GPU code for running sparse 2, 3 and 4 dimensional CNNs is available at:

`https://github.com/btgraham/SparseConvNet`

The world we live in is three dimensional, and time can also be thought of as an extra dimension, so there are a large number of possible applications for three and even four dimensional CNNs. Figure 1 shows what happens to sparse 3D data as it passes though a CNN. In this paper I apply CNNs to a variety of sparse 3D datasets.

When applying CNNs to sparse data, it may be better to use small convolutional filters, as they do a better job of preserving sparsity in the computationally expensive lower layers of the network. To reduce the size of the filters, we have experimented with changing the underlying graph. See Figure 2.

Figure 3 shows a variety of objects from the SHREC2015 Non-rigid 3D Shape Retrieval dataset, each stored as a mesh of triangles in the OFF-file format. The dataset contains 1200 exemplars split evenly between 50 classes (aliens, ants, armadillo, ...). The dataset was intended to be used for unsupervised learning, but as CNNs are most often used for supervised learning, we used 6-fold cross-validation to measure the ability of our 3D CNNs to learn shapes. To stop the dataset being too easy, we randomly rotated the objects during training and testing. This is to force the CNN to truly learn to recognize shape, and not rely on some classes of objects tending to have a certain orientation. We tested a variety of network architectures to explore the trade-off between speed and accuracy.

Figure 4 shows an image from the Recognizing Human Actions video dataset. Taking differences between successive frame converts the dataset to a collection of sparse 2+1 dimensional objects. We also experimented with the more complicated UCF101 video dataset.

[1] Ben Graham. Spatially-sparse convolutional neural networks. 2014. URL `http://arxiv.org/abs/1409.6070`.

[2] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(1):221–231, January 2013. ISSN 0162-8828. doi: 10.1109/TPAMI.2012.59. URL `http://dx.doi.org/10.1109/TPAMI.2012.59`.

[3] D. Maturana and S. Scherer. 3D Convolutional Neural Networks for Landing Zone Detection from LiDAR. In *ICRA*, 2015.

[4] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

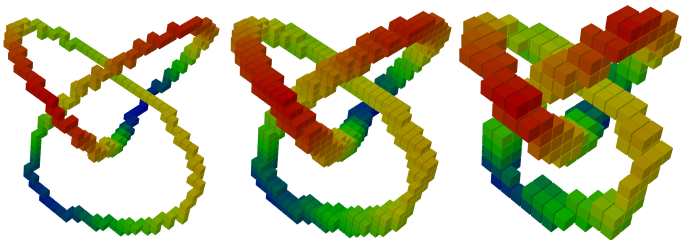

Figure 1: Left to right: A trefoil knot has been drawn in the cubic lattice; these are the input layer's active sites. Applying a $2 \times 2 \times 2$ convolution, the number of active (i.e. non-zero) sites increases. Applying a $2 \times 2 \times 2$ pooling operation reduces the scale, which tends to decrease the number of active sites.

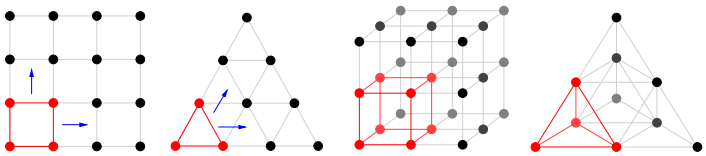

Figure 2: Convolutional filter shapes for different lattices: (i) A 4×4 square grid with a 2×2 convolutional filter. (ii) A triangular grid with size 4, and a triangular filter with size 2. (iii) A $3 \times 3 \times 3$ cubic grid, and a $2 \times 2 \times 2$ filter. (iv) A tetrahedral grid with size 3, and a filter of size 2.

In d-dimensions, the smallest practical filter for the triangular/tetrhedral/hypertetrahedral lattice contains $d+1$ vertices, while the smallest practical filter for the square/cubic/hypercubic lattice contains 2^d vertices. Thus, the higher the dimension, the greater the potential benefit in terms of efficiency from considering non-square lattices.

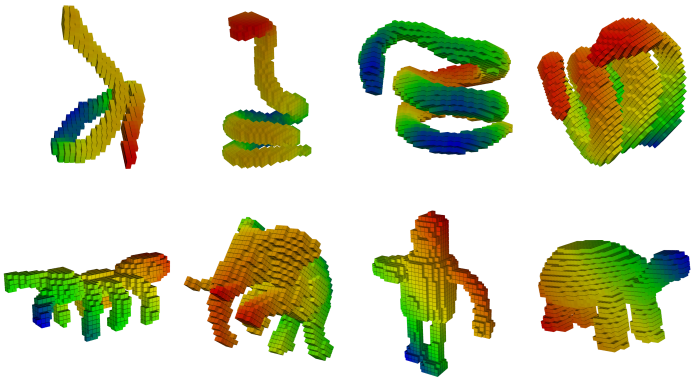

Figure 3: Items from a 3D object dataset, embedded into a $40 \times 40 \times 40$ cubic grid. Top row: four items from the snake class. Bottom row: an ant, an elephant, a robot and a tortoise.

Figure 4: Left: a frame from a video from the Recognizing Human Actions video dataset. Right: the difference between that frame and the previous frame, with thresholding applied to increase sparsity. Stacking the frame differences produces a sparse 2+1 dimensional space-time object.