Compressed Volume Rendering using Deep Learning

Somay Jain, Wesley Griffin, Afzal Godil, Jeffrey W. Bullard, Judith Terrill and Amitabh Varshney

Abstract—Scientific simulations often generate a large amount of multivariate time varying volumetric data. Visualizing these volumes is absolutely essential for understanding the underlying scientific processes which generate this data. In this paper, we present a method to obtain a data-driven compact representation of the given volumes using a deep convolutional autoencoder network. We show that the autoencoder learns high level hierarchical features, giving insights about the the distribution of the underlying data. Moreover, the compact representation has surprisingly low storage requirements which enables it to fit on the Graphical Processing Unit (GPU) memory. The compact representation for a given time step is efficiently decompressed using GPUs to achieve interactive speeds for rendering and navigating large time varying datasets. Finally, the compact representation can also be used to transmit very large volumes over bandwidth sensitive networks. We show that our proposed compact representation takes only 7% of the original memory and reconstructs the original volume with minimal error.

Index Terms—Convolutional autoencoder neural network, Volume rendering

1 Introduction

Scientific simulations often carried out using supercomputers generate a large amount of volumetric data, which spans hundreds of time steps, each having millions of voxels containing large scalar and vector fields. Such multivariate time varying datasets can take several gigabytes or even terabytes of space. The ability to visualize this immense amount of information is absolutely essential to interpret, analyze and gain insights about the underlying scientific processes which generate this data.

Sharing and visualization of immense datasets are key to facilitating new scientific discoveries. However, commodity systems available to many researchers around the world typically have insufficient memory and graphics hardware to handle and visualize such large datasets. Though direct volume rendering techniques are good for visualizing small datasets, they cannot feasibly process very large datasets generated using supercomputers. These obstacles could be overcome by developing a compressed representation of the data and the ability to visualize directly from that compressed representation.

Recent advances in deep learning have proved to be very useful in areas like computer vision [1], computational linguistics [2] and audio processing [3] with the goal of learning the underlying data distribution [4], [5]. Though the use of deep learning in computer graphics is still limited, the ability to capture the underlying distribution of the given volume has been used for efficient interactive volume rendering [6], [7], [8].

Time varying volumes often contain implicit structures in the data which repeat in space and time. In this work, we propose a novel method to perform compressed volume rendering of multivariate time varying volumes. We use a convolutional autoencoder neural network to learn hierarchical features which capture the implicit repeating structures present in the data. A compressed representation of the data is generated using these high level features. During volume rendering, a lossy reconstruction of the original data is obtained by a non linear combination of the compressed representation and the learned features.

The main contributions of this paper are as follows -

1) We devise a method to learn a data-driven lossy compressed representation of time-varying multivariate volumes. The lossy representation is sufficient for data exploration tasks like volume rendering.
2) We show that our model learns high level hierarchical features, giving insights about the underlying data distribution.
3) We provide a real time GPU-based method to decompress the compressed representation on the fly.
4) We propose that the above compression technique can be used to transfer and store very large scientific datasets generated by systems with high computational capacity.

The remainder of the paper is organized as follows: We review the related work in section 2. In section 3, we describe our method in detail, including preprocessing the data, learning the compressed representation and real time rendering. Results and analysis on various multi-variate time varying datasets are described in section 4. Finally, we conclude and discuss future work in section 5.

- *S. Jain is with the Department of Computer Science, University of Maryland, College Park, MD 20742.*
  E-mail: somay@cs.umd.edu
- *A. Varshney is with the University of Maryland Institute for Advanced Computer Studies, College Park, MD 20742.*
  E-mail: varshney@umiacs.umd.edu
- *W. Griffin, A. Godil, J. Bullard and J. Terrill are with National Institute of Standards and Technology, Gaithersburg, MD 20899.*
  E-mail: {wesley.griffin, afzal.godil, jeffrey.bullard}@nist.gov

2 Related Work

Recent years have witnessed a very rapid increase in computational power, with availability of faster processors and cheaper storage. However, because of a similar technological development in supercomputers, there is a disparity between the size of generated data and the size of data which can be efficiently processed and visualized on a commodity system. In response to this disparity, several techniques have been proposed for performing direct volume rendering from compressed data. Most of these techniques approximate the original data as a weighted linear combination of elementary bases signals. The bases are either analytically determined from pre-defined models, or they are learned individually for a given dataset. Rodrguez et al. [9] summarize a range of GPU-based compressed volume rendering techniques.

Dunne et al. [10] used Discrete Fourier Transform (DFT) to compress the dataset in the Fourier domain, represented by sine and cosine signals. Though DFT renders directly in the compression domain, it is restricted to rendering a projection in the direction perpendicular to the slice and does not allow the use of transfer functions, shading models and perspective projections. Moreover, the DFT representation is unable to localize spatial structures.

Muraki et al. [11] introduced Discrete Wavelet Transform (DWT), which transforms the data into frequency domain while maintaining the spatial domain. DWT processes the data using low-pass and high-pass filters. The low-pass filters provide a coarse approximation while the high-pass filters provide a detailed approximation of the volume. Westermann et al. [12] showed that DWT is especially good for block-based multi resolution rendering. Guthe et al. [13] used a hierarchical wavelet representation to render large datasets at interactive speeds. DFT and DWT rely on analytically compressing the data using pre-defined models, regardless of the structures within the data. Hence, they are not able to capture global patterns which occur especially in time varying datasets.

Several approaches are able to learn the bases needed for compression individually for each dataset. Fout et al. [6] use the linear Karhunen-Loeve Transform (KLT) to remove redundancies in the data by estimating the covariances and eigenvectors. KLT is closely related to Principal Component Analysis (PCA), which projects the input data in a lower dimensional subspace using a linear combination of uncorrelated bases. The disadvantage with KLT is that it does not have fast forward and inverse transforms.

Kolda et al. [7] describe Tensor Approximation (TA) as approximating a multi-dimensional input dataset (*i.e.*, a tensor) as a sum of rank-one tensors or as a product of a core tensor and matrices for each dimension (Tucker decomposition). These are higher order generalizations of the Singular Value Decomposition (SVD) and Principal Component Analysis (PCA). Suter et al. [8] show that TA based methods capture repeating structures in the data better than wavelet transforms, thus being more suitable for interactive large volume visualization.

Deep learning models have proven to be very useful for learning the underlying distribution of the data [4], [5]. There also has been work on understanding the expressive power of deep learning models [14] [15]. Cohen et al. [15] look at it from the perspective of tensor decomposition. They show that a shallow neural network corresponds to a rank-one tensor decomposition, whereas a deep neural network corresponds to a Hierarchical Tucker decomposition. Thus, a deep neural network is a hierarchical representation of the tensor decomposition and has a richer representational capacity than Tensor Approximation (TA). Moreover, deep convolutional autoencoder networks are known to learn features invariant to translations, rotations and deformations [16]. Decomposing the data as a hierarchical combination of robust features allows us to store a compact representation of the data.

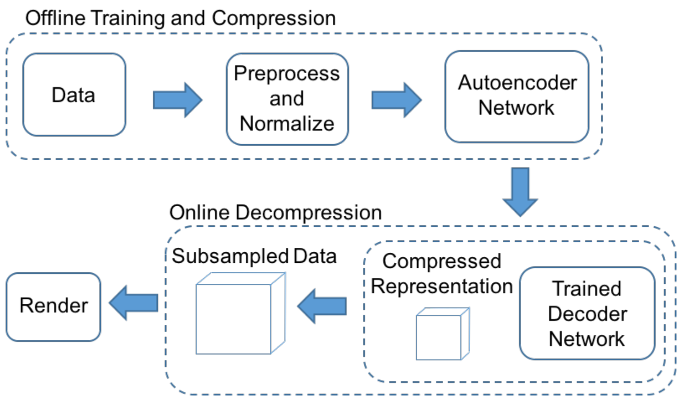


Fig. 1: Our proposed framework first preprocess the data and trains the autoencoder network to learn a high-level compressed representation as an offline process. It then uses the compressed representation and trained decoder network to generate the subsampled data on the fly, which is visualized using ray casting.

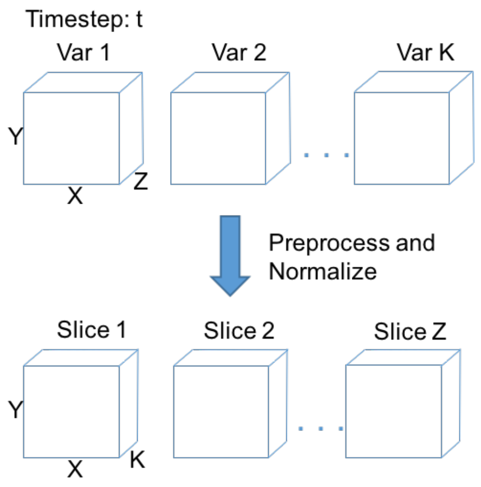


Fig. 2: Preprocessing the data: Normalizing and arranging the data for each time step as multivariate slices

3 Approach

The proposed framework describes a method for rendering time varying multivariate volumes from learned compressed representation as summarized in Figure 1. We first

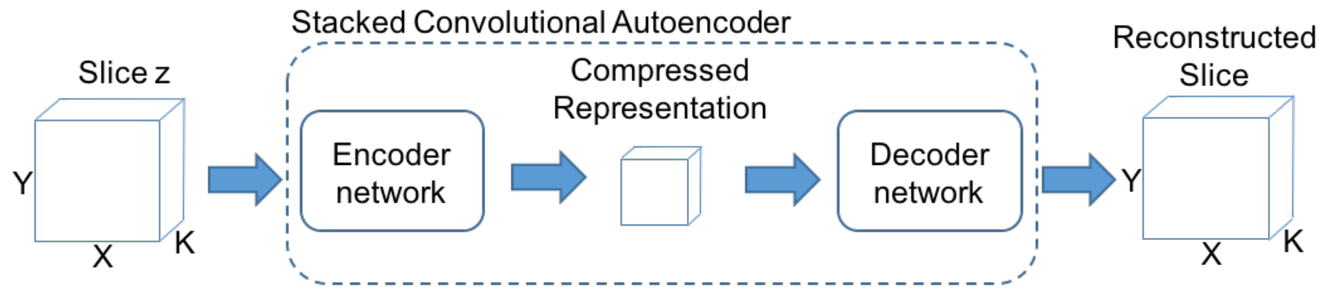


Fig. 3: Schematic representation of the autoencoder network trained using backpropogation.

preprocess the data into multivariate slices along the z-axis and then train a stacked convolutional autoencoder network to capture the high level features in the data. After training, the autoencoder network is split into encoding and decoding networks. The output layer of the encoding network gives a compressed representation of the data, which is computed as a nonlinear combination of input data and hierarchical features learned by the network. The decoder network takes this compressed representation and generates a lossy reconstruction of the input data. The volume renderer uses the decoder network to decompress the volume corresponding to a time step on the fly as needed.

3.1 Preprocessing data

The given volume can be represented as a multidimensional array of size $X \times Y \times Z \times K \times T$, where X, Y, Z are the dimensions of the volume, K is the dimension of the multivariate field represented by each voxel and T is the number of time steps of the time-varying volume. Figure 2 gives an overview of the preprocessing step. We first normalize the data between 0 and 1 for each multivariate dimension across all time steps. We then arrange the data as multivariate slices along the z-axis. This is done so that the multivariate field acts as a channel to the autoencoder network.

3.2 Autoencoder Architecture

Our proposed stacked convolutional autoencoder architecture consists of convolutional, activation, max-pooling and upsampling layers. Figure 3 shows a schematic representation of the autoencoder. The architecture is composed of an encoder network, which converts the input into a compressed representation, and a decoder network, which reconstructs the input with minimal error. The encoder and decoder network are jointly trained using backpropogation. Once trained, the encoder and decoder networks are separated and used for compressing and decompressing the data, respectively.

3.2.1 Convolutional Layers

The convolutional layer consists of trainable filters (or kernels) $W_{p,q}$ which are convolved with the input volume U of size $x \times y \times p$ to generate a volume V of size $x \times y \times q$, where x and y correspond to the spatial dimensions, p and q represent the number of channels (or feature maps) in U and V respectively. The i-th channel V_i of the output volume is calculated as

$$V_i = \sum_{j=1}^{p} W_{j,q} * U_j + b_i \tag{1}$$

where b_i represents the trainable bias associated with the channel V_i. The size of filters is kept constant 3×3 in this work. While the receptive field captured by these filters is small, multiple convolution layers are stacked one after the other to capture a larger receptive field with fewer trainable parameters [17]. Two convolution layers with 3×3 filters has an effective receptive field of 5×5 and three such layers have an effective receptive field of 7×7.

3.2.2 Activation Layers

The activation layers apply a nonlinear activation function f to all elements v_i of the convolutional layer output V. We use Rectified Linear Units (ReLUs) to propagate only the positive inputs to the next layer for all convolutional layers except the output layer:

$$f(v_i) = \max(0, v_i) \tag{2}$$

We apply the sigmoid activation function to the output convolutional layer of the autoencoder network to obtain the output between 0 and 1.

$$f(v_i) = \frac{1}{1 + e^{-v_i}} \tag{3}$$

3.2.3 Max-pooling Layers

Max-pooling layers apply a max filter to non-overlapping sub-regions of the input so as to downsample the input and reduce the number of parameters in subsequent layers of the encoder network. These layers do not contain any trainable parameters.

3.2.4 Upsampling Layers

Upsampling layers simply expands the spatial dimensions x and y of the input volume U of size $x \times y \times p$ by s_x and s_y respectively to generate a volume V of size $(s_x x) \times (s_y y) \times p$. These layers do not contain any trainable parameters and are used in the decoder network to reconstruct the volume of the original size.

3.2.5 Training

Table 1 shows the layer architecture of the autoencoder network. The network is trained with multivariate slices from all time steps, obtained after the preprocessing step as outlined in section 3.1. The output of the network is

Layer	Input Channels	Output Channels	Stride	Filter Size
Convolutional	1	64	1	3x3
Convolutional	64	64	1	3x3
Max Pooling	64	64	2	2x2
Convolutional	64	64	1	3x3
Convolutional	64	32	1	3x3
Max Pooling	32	32	2	2x2
Convolutional	32	16	1	3x3
Convolutional	16	8	1	3x3
Max Pooling	8	8	2	2x2
Convolutional^E	8	4	1	3x3
Convolutional	4	8	1	3x3
Convolutional	8	16	1	3x3
Convolutional	16	32	1	3x3
Up Sampling	32	32	2	2x2
Convolutional	32	64	1	3x3
Convolutional	64	64	1	3x3
Up Sampling	64	64	2	2x2
Convolutional	64	64	1	3x3
Convolutional	64	64	1	3x3
Up Sampling	64	64	2	2x2
Convolutional^D	64	1	1	3x3

TABLE 1: The architecture of our proposed convolutional autoencoder. The output of the Convolutional^E layer gives the compressed representation of the input slice. Output of Convolutional^D layer gives the reconstructed slice.

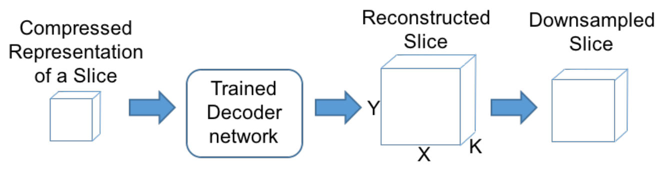


Fig. 4: Decompression at a specific resolution from the learned compressed representation and trained decoder network.

a reconstruction of the input slice. We minimize the cross entropy between the input slice and the reconstructed slice using the ADADELTA solver [18] with a momentum of 0.95. We train the network by randomly sampling slices from the input data, until the mean squared error between the input volume and the reconstructed volume stops to decrease. The number of iterations required during training depends on the complexity of the dataset and the desired quality of reconstruction.

3.3 Decompression

Once the autoencoder network is trained, the compressed representation for each multivariate slice and the decoder network are retained. Decompressing a slice at original resolution is done by feeding the compressed slice through the decoder network. This involves a series of basic convolution, max and upsampling operations. The process of decompression is done per slice on the fly whenever required so that it only requires additional memory corresponding to one reconstructed slice at full resolution, as outlined in Figure 4.

Our proposed architecture enforces that the learned representation is small enough to reside on the memory and the decoder network is simple, consisting of basic operations which can be carried out on the GPU. This enables interactive rendering of very large time-varying multivariate volumes.

3.4 Volume rendering

While rendering, the volume for the required time step is decompressed with the GPU, using the decoder network on the fly. A coarser volume is first obtained by decompressing subsampled slices along the z-axis. The coarser volume is continuously refined by decompressing the remaining slices while the coarser volume is being rendered. Decompression is performed until a finer volume is obtained, after which rendering is done directly from the finer volume. This is done to facilitate real time switching between time steps while exploring the time varying volume.

We use the ray casting algorithm [19] for visualizing the volume. For each pixel in the output image, we cast a ray into the volume along the viewing direction. The color for the target pixel is computed by compositing the colors of the sampled points along the ray. The mapping from voxel intensity to color and opacity is given by the user defined transfer function.

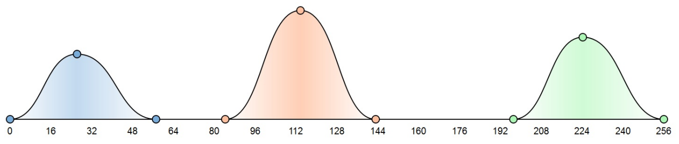


Fig. 5: User interface for specifying the transfer function used in volume rendering.

Figure 5 shows the user interface for specifying the transfer function. The x-axis represents the intensity of the voxel scaled between 0 to 255 and y-axis represents the opacity. Users can edit the transfer function by adding, removing, or moving the control points. Users can also assign a unique color and opacity to each control point. The color and opacity are linearly interpolated between adjacent control points. Changes in the transfer function are dynamically reflected in volume rendering in real time.

4 Results

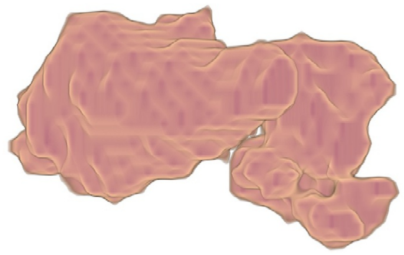
4.1 Implementation

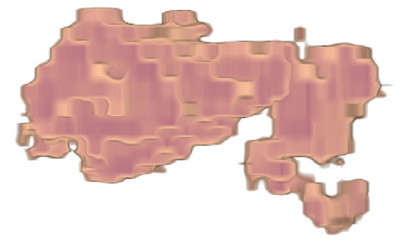
We implement our deep learning compression and decompression networks using the Keras framework [20]. The GPU-based rendering from compressed representation is implemented using cuDNN [21], OpenGL and CUDA. Training the autoencoder network is done as an offline process on a system with Intel Xeon 2.6 GHz CPU and a NVIDIA Tesla K80 GPU. Volume rendering is run on a Intel Xeon 2.1 GHz CPU with a NVIDIA GTX 1080 GPU.

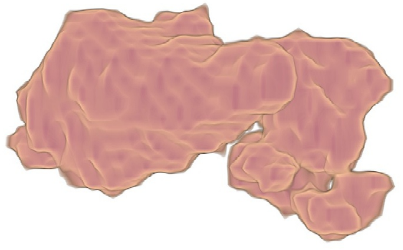
4.2 Datasets

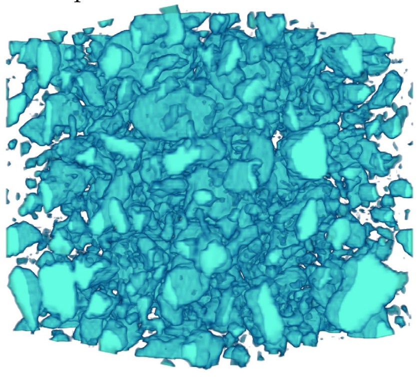
The datasets were generated on the Texas Advanced Computing Center computer, Stampede, through an NSF XSEDE grant [22]. The size of the time varying multivariate datasets used are outlined in Table 2.

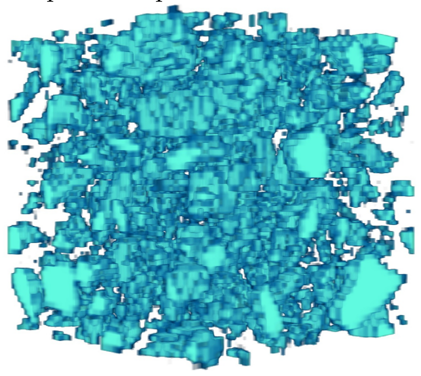
The datasets in Table 2 all are representations of micrometer-sized Ca_3SiO_5 particles suspended in water to

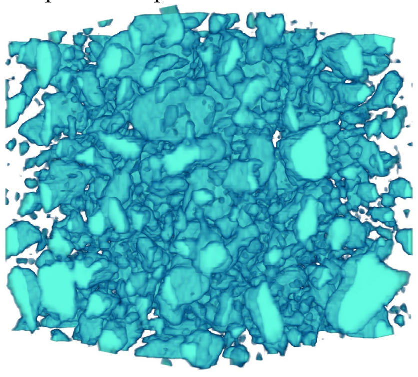

(a) Dataset1: Rendering from uncompressed volume

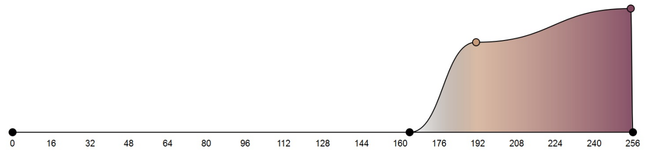

(b) Dataset1: Rendering from wavelets compressed representation


(c) Dataset1: Rendering from learned compressed representation

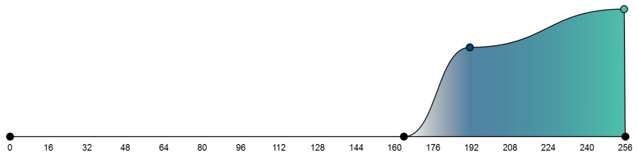

(d) Dataset2: Rendering from uncompressed volume


(e) Dataset2: Rendering from wavelets compressed representation


(f) Dataset2: Rendering from learned compressed representation



(g) Transfer function used for Dataset1



(h) Transfer function used for Dataset2

Fig. 6: Comparison of volume rendering of Ca_3SiO_5 concentration from Dataset 1 and 2 at an intermediate time step. The compressed representations (sub-figures b, c, e and f) take 7 % of the total memory. It can be seen that rendering from our learnt representation (sub-figures c and f) is very similar to the uncompressed volume (sub-figures a and d), whereas rendering from wavelets compressed representation (sub-figures b and e) results in a very coarse reconstruction. (g, h) show the transfer function used.

Name	Dimensions of the volume	Elements per voxel	No of timesteps	Size
Dataset1	52 x 52 x 37	39	88	2.56GB
Dataset2	100 x 100 x 100	39	20	5.81GB
Dataset3	100 x 100 x 100	39	196	56.95GB

TABLE 2: Description of the datasets used

initiate dissolution and precipitation reactions. Ca_3SiO_5 is the majority mineral component of portland cement, and its reaction in water governs the early-time heat release, setting, and strength development in concrete. Therefore, Ca_3SiO_5 is often used as a convenient proxy in experimental and computational investigations of portland cement hydration.

The dataset labeled "Dataset1" in Table 2 is a collection of nine micrometer-sized particles of Ca_3SiO_5 affixed to a tungsten needle, which were subsequently submerged in water to initiate hydration reactions. The system was constructed from X-ray microtomography scans of an actual system. The hydration behavior of this system and comparisons to computer simulations have been reported recently [23], [24]. Datasets 2 and 3 were created by randomly parking Ca_3SiO_5 particles to a volume fraction of 0.38, which is typical of the solid volume fraction in concrete binders. The particle size distribution is unimodal with a range of $[0.125\,\mu m, 75\,\mu m]$ and a mode of $22\,\mu m$, which is typical of portland cement powder. The shapes were reproduced by spherical harmonic modeling of X-ray microtomography scans of thousands of particles of a reference cement [25].

4.3 Quality of Reconstruction

Figure 6 shows rendering of Ca_3SiO_5 concentration from Dataset 2 at an intermediate timestep. It shows the comparison between rendering from the wavelets compressed representation and our autoencoder learnt representation. Both representations use 7 % of the total memory of the dataset. It can be seen that rendering from the wavelets compressed representation gives a very coarse reconstruction of the original volume, with a lot of visual artifacts. On the other hand, rendering from our learnt representation gives a very close approximation of the original volume with no visible artifacts.

The quantitative quality of the reconstructed volume is measured using the mean squared error (MSE) between the

Model	Dataset1		Dataset2	
	MSE	PSNR	MSE	PSNR
Daubechies1 Wavelet	0.0026	25.850	0.0454	13.432
Daubechies2 Wavelet	0.0017	27.695	0.0360	14.430
Daubechies3 Wavelet	0.0016	27.958	0.0338	14.707
Discrete Mayer Wavelet	0.0015	28.239	0.0320	14.943
Biorthogonal 9/3 Wavelet	0.0024	26.197	0.0219	16.588
CNN Autoencoder	**0.000061**	**42.125**	**0.0049**	**23.098**

TABLE 3: Mean Squared Error (MSE) and Peak Signal to Noise Ratio (PSNR) of reconstruction from 7 % data (14.28:1 compression ratio) for various models

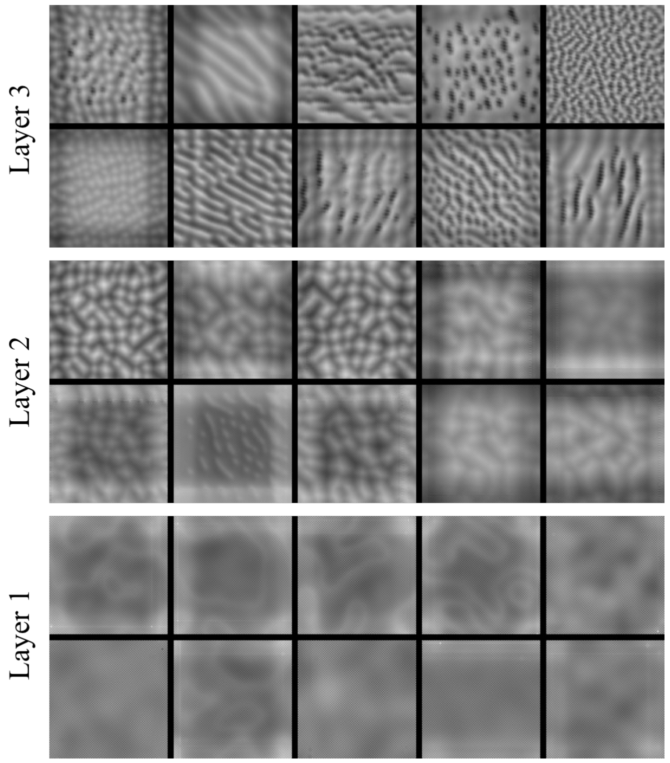


Fig. 7: Visualization of filters learnt in the first three layers of the autoencoder network. Small images represent the input which maximizes the output of the visualized filter, depicting the features captured by the filter. It can be seen that the features are hierarchical and their complexity increases with depth of the layer. The first layer learns basic features, which are combined to form textures in the second layer, which combine to form complex patterns in the third layer. Best viewed while zoomed in.

original and the reconstructed value. Another metric used is the Peak Signal to Noise Ratio (PSNR), which is calculated as

$$PSNR = 10 \ \ log_{10}\left(\frac{(MAX_I)^2}{MSE}\right)$$

where MAX_I is the maximum possible intensity of the voxels present in the volume.

Table 3 show a quantitative comparison of reconstruction between our proposed approach and wavelet based approaches. The compression ratio of the compressed representation is 14.28:1. Our proposed approach gives much higher quality of reconstruction with very low storage requirements.

4.4 Analysis of Learned Filters

Our proposed approach learns a hierarchical representation of the structures present in the volume. To interpret the representation learned by our convolutional autoencoder network, we visualize the convolution filters at each layer using regularized gradient ascent [26]. We start with a random input and iteratively modify it by taking an ascent step in the direction of the gradient of the filter. This generates an input which maximizes the output of the filter we wish to visualize.

Figure 7 shows the visualization of filters from selected filters of the first three layers of the autoencoder network. Each small image corresponds to the visualization of one filter of the network. The features learned are hierarchical and their complexity increases with the depth of the layers. The first layer of the network learns basic features. These basic features are then combined to form simple textures in the second layer, which eventually combine to form complex structures and patterns in the third layer. The high level features captured by the deeper layers also give an insight about the underlying distribution of the data.

5 Conclusion

Scientific simulations carried out on systems with high computational capacity often generate large amount of multivariate time varying volumetric data. Visualizing this immense amount of data in real time on commodity systems is a huge challenge. In this work, we apply a deep learning based approach to capture hierarchical features in the data and use them as learned bases to form a compact representation of the input data. The compact representation has surprisingly low storage requirements and a reconstruction of the original data is obtained from the compact representation with very low reconstruction error. We show that the quality of compression outperforms the classical wavelets based compression methods. While rendering, we store the compact representation for all time steps on the GPU memory and reconstruct the original data on the fly as needed. This facilitates real time rendering and exploration of the time varying dataset.

In the future, one could investigate in learning similar models using high performance distributed computing, so that the model could be learned while the simulation is running.

Acknowledgments

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1053575.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[2] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[3] H. Lee, P. Pham, Y. Largman, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in neural information processing systems*, 2009, pp. 1096–1104.

[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[5] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[6] N. Fout and K. L. Ma, "Transform coding for hardware-accelerated volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1600–1607, Nov 2007.

[7] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[8] S. K. Suter, C. P. E. Zollikofer, and R. Pajarola, "Application of Tensor Approximation to Multiscale Volume Feature Representations," in *Vision, Modeling, and Visualization (2010)*, R. Koch, A. Kolb, and C. Rezk-Salama, Eds. The Eurographics Association, 2010.

[9] M. B. Rodrguez, E. Gobbetti, J. A. I. Guitin, M. Makhinya, F. Marton, R. Pajarola, and S. K. Suter, "A survey of compressed gpu-based direct volume rendering," 2013.

[10] S. Dunne, S. Napel, and B. Rutt, "Fast reprojection of volume data," in *[1990] Proceedings of the First Conference on Visualization in Biomedical Computing*, May 1990, pp. 11–18.

[11] S. Muraki, "Volume data and wavelet transforms," *IEEE Comput. Graph. Appl.*, vol. 13, no. 4, pp. 50–56, Jul. 1993. [Online]. Available: http://dx.doi.org/10.1109/38.219451

[12] R. Westermann, "A multiresolution framework for volume rendering," in *Proceedings of the 1994 Symposium on Volume Visualization*, ser. VVS '94. New York, NY, USA: ACM, 1994, pp. 51–58. [Online]. Available: http://doi.acm.org/10.1145/197938.197963

[13] S. Guthe, M. Wand, J. Gonser, and W. Strasser, "Interactive rendering of large volume data sets," in *Proceedings of the Conference on Visualization '02*, ser. VIS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 53–60. [Online]. Available: http://dl.acm.org/citation.cfm?id=602099.602106

[14] Y. Bengio and O. Delalleau, "On the expressive power of deep architectures," in *International Conference on Algorithmic Learning Theory*. Springer, 2011, pp. 18–36.

[15] N. Cohen, O. Sharir, and A. Shashua, "On the expressive power of deep learning: A tensor analysis," *arXiv preprint arXiv:1509.05009*, vol. 554, 2015.

[16] I. J. Goodfellow, Q. V. Le, A. M. Saxe, H. Lee, and A. Y. Ng, "Measuring invariances in deep networks," in *Proceedings of the 22Nd International Conference on Neural Information Processing Systems*, ser. NIPS'09. USA: Curran Associates Inc., 2009, pp. 646–654. [Online]. Available: http://dl.acm.org/citation.cfm?id=2984093.2984166

[17] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[18] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[19] S. D. Roth, "Ray casting for modeling solids," *Computer graphics and image processing*, vol. 18, no. 2, pp. 109–144, 1982.

[20] F. Chollet, "Keras," https://github.com/fchollet/keras, 2015.

[21] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014. [Online]. Available: http://arxiv.org/abs/1410.0759

[22] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, "Xsede: Accelerating scientific discovery," *Computing in Science Engineering*, vol. 16, no. 5, pp. 62–74, Sept 2014.

[23] Q. Hu, M. Aboustait, T. Kim, M. T. Ley, J. C. Hanan, J. Bullard, R. Winarski, and V. Rose, "Direct three-dimensional observation of the microstructure and chemistry of c_3s hydration," *Cement and Concrete Research*, vol. 88, pp. 157 – 169, 2016.

[24] J. W. Bullard, J. G. Hagedorn, M. T. Ley, Q. Hu, W. Griffin, and J. E. Terrill, "A critical comparison of 3D experiments and simulations of tricalcium silicate hydration," *J. Am. Ceram. Soc.*, p. submitted, 2017.

[25] E. J. Garboczi and J. W. Bullard, "Shape analysis of a reference cement," *Cement and Concrete Research*, vol. 34, no. 10, pp. 1933–1937, 2004.

[26] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.