Neurocomputing 174 (2016) 988–998

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

An efficient and effective convolutional auto-encoder extreme learning machine network for 3d feature learning

Yueqing Wang [a,b,*], Zhige Xie [b], Kai Xu [b], Yong Dou [a,b], Yuanwu Lei [b]

[a] National Laboratory for Parallel and Distributed Processing, National University of Defense Technology, Changsha, China
[b] College of Computer, National University of Defense Technology, Changsha, China

ARTICLE INFO

Article history:
Received 2 September 2015
Received in revised form
5 October 2015
Accepted 11 October 2015
Communicated by G.-B. Huang
Available online 19 October 2015

Keywords:
Convolutional
Extreme learning machine
Auto-encoder
Feature learning

ABSTRACT

3D shape features play a crucial role in graphics applications, such as 3D shape matching, recognition, and retrieval. Various 3D shape descriptors have been developed over the last two decades; however, existing descriptors are handcrafted features that are labor-intensively designed and cannot extract discriminative information for a large set of data. In this paper, we propose a rapid 3D feature learning method, namely, a convolutional auto-encoder extreme learning machine (CAE-ELM) that combines the advantages of the convolutional neuron network, auto-encoder, and extreme learning machine (ELM). This method performs better and faster than other methods. In addition, we define a novel architecture based on CAE-ELM. The architecture accepts two types of 3D shape representation, namely, voxel data and signed distance field data (SDF), as inputs to extract the global and local features of 3D shapes. Voxel data describe structural information, whereas SDF data contain details on 3D shapes. Moreover, the proposed CAE-ELM can be used in practical graphics applications, such as 3D shape completion. Experiments show that the features extracted by CAE-ELM are superior to existing hand-crafted features and other deep learning methods or ELM models. Moreover, the classification accuracy of the proposed architecture is superior to that of other methods on ModelNet10 (91.4%) and ModelNet40 (84.35%). The training process also runs faster than existing deep learning methods by approximately two orders of magnitude.

1. Introduction

3D shape feature extraction is a vital issue covered in the high-level understanding of 3D shapes. Extensive efforts have been exerted to solve this important problem with the aid of recent advances on deep learning techniques. Existing feature extraction approaches based on deep learning can be broadly categorized as semi-automatic and fully-automatic methods.

In semi-automatic methods such as [1,2], researchers first extract several popular hand-crafted features from input 3D shapes and then utilize deep learning methods to combine these features further. This category of methods relies strongly on the adopted human-designed features. The extraction of these features consumes much time; hence, these methods cannot handle large-scale 3D datasets.

Numerous fully automatic deep learning methods have been proposed recently, such as convolutional deep belief network (CDBN) [3], auto-encoder (AE) [4], deep Boltzmann machines [5], convolutional neuron network (CNN) [6], and stacked local convolutional AE [7] approaches. These techniques are utilized to learn 3D features given the feature learning capability of these methods. In addition, these methods were first proposed for 2D image classification tasks.

3D shapes with reasonable resolutions have the same dimensions as high-resolution images. Thus, training deep networks on large-scale 3D datasets is time consuming. Furthermore, mastering this category of feature learning methods consumes time because of the black-box property of the deep learning method. Most of these deep learning methods convert 3D shapes into 2D representations for input [7–9]; thus, much of the 3D geometry information of 3D shapes is lost. Several works [3,10] attempt to apply 3D cubes, such as the volumetric representations of 3D shapes, as inputs. However, the training processes of these works are time consuming because of the additional dimension of input data. Therefore, the input resolution of these methods is limited.

To overcome the shortcomings of the existing methods, we propose a novel 3D shape feature extraction method called convolutional AE extreme learning machine (CAE-ELM) in this paper. This approach combines the advantages of CNN, AE, and extreme learning machine (ELM). AE is a typical unsupervised learning

* Corresponding author.
E-mail address: yqwang2013@163.com (Y. Wang).

http://dx.doi.org/10.1016/j.neucom.2015.10.035
0925-2312/

algorithm that can extract good features without supervised labels. However, the AE network is fully connected; thus, additional parameters must be learned. CNN restricts the connections between the hidden layer and the input layer through locally connected networks. Nevertheless, this network is an extensive computational method that is used with 3D shape datasets because of its convolutional operation. To reduce computational complexity, ELM [11] is often considered for its high efficiency and effectiveness.

Additionally, different input representations exert varied effects. For example, voxel data describe the structural information of 3D shapes because these data are expressed only as 0 and 1, which indicate that the voxel is outside and inside the mesh surface, respectively. Signed distance field (SDF) data are represented as a grid sampling of the minimum distance to the surface of an object that is represented as a polygonal model. The convention of applying negative and positive values within and outside the object, respectively, is frequently applied; thus, additional 3D shape details can be derived. To extract the global and local features collectively, we define a novel architecture that accepts both voxel and SDF as inputs. By combining these two types of data, our architecture can classify 3D shapes effectively.

The proposed CAE-ELM can also be used in practical graphics applications, such as in 3D shape completion. Optical acquisition devices often generate incomplete 3D shape data because of occlusion and unfavorable surface reflectance properties. These incomplete 3D shapes are challenging to repair; to fix incomplete data, we compare the features of broken and complete shapes before the CAE-ELM classifier as well as obtain the broken locations and values. Although the completion results are imperfect, CAE-ELM serves as a new approach to solve this problem.

The contributions of our approach are summarized as follows:

(1) *CAE-ELM*: We propose a new ELM-based designed network that performs well and learns quickly. To the best of our knowledge, our proposed model is the first to combine the advantages of CNN, AE, and ELM to learn the features of 3D shapes. This method has been used in practical graphics applications. We provide the source code[1] so that researchers can master it in a short time.
(2) *Increased classification accuracy*: The classification accuracy of the designed architecture is higher than that of other methods [10,12,13,9] on ModelNet10 (91.41%) and ModelNet40 (84.35%).
(3) *3D shape completion*: CAE-ELM can repair a broken 3D shape by using the features before the classifier.
(4) *Rapid 3D shape feature extraction*: Our method runs faster than existing deep learning methods by approximately two orders of magnitude, thus facilitating large-scale 3D shape analysis.

The experiment results show that the features learned by CAE-ELM significantly outperform hand-crafted features and other deep learning methods in terms of 3D shape classification. CAE-ELM can also repair the broken locations of 3D shapes with learned features for 3D shape completion. Furthermore, our method is efficient and easy-to-implement; thus, it is practical for real 3D applications.

2. Related work

2.1. 3D shape descriptors

3D shape descriptors play a crucial role in graphics applications such as 3D shape matching, recognition, and retrieval [14–17].

A variety of 3D shape descriptors have been developed during the last two decades [18,13,19,15]. Existing 3D descriptors are hand-crafted features which are labor-intensively designed and are unable to extract discriminative information from the data. Instead, we learn shape features from 3D shapes using automatically feature learning method.

2.2. 3D feature learning via deep learning

Researchers have successfully built deep models, such as convolutional neural network (CNN) [20], deep autoencoder networks [21], deep belief nets (DBN) [22] and extreme learning machine (ELM) [23] and etc., to automatically extract features with the superior discriminatory power for 2D image and shape representation in computer vision and machine learning [24]. A few very recent works attempt to learn 3D shape features via deep learning methods.

Zhang et al. [25] use ELM to determine an optimal fabrication in 3D printing considering a perceptual model. Wu et al. [3] use voxelization of 3D meshes as network's input and adopt 3D deep belief nets (DBN) [22] as their networks. Their work obtains good results on a subset of Princeton ModelNet [3]. However, their method is timing consuming and discards the pooling operations in CDBN, which makes their network fail to handle shape rotation invariance. As a result, they have to manually align all the input meshes into the same direction, in order to avoid uncertainty in extracting their features. Zhu et al. [4] use autoencoder to learn a 3D shape feature based on the depth images. However, they treat 2.5D depth images as traditional 2D images and can only get the global feature, which make their method have to combine with hand-crafted 2D image features (SIFT) to finish the 3D shape classification task. Xie et al. [9] propose Multi-View Deep Extreme Learning Machine (MVD-ELM) which adopts the multi-view depth image representation can achieve fast and quality projective feature learning for 3D shapes. However, as mentioned before, using 2.5D depth images as the input of network will lose 3D geometry and structure information of 3D shapes, and further influences the classification accuracy. Our method can handle large scale of 3D shapes with large rotation and geometry invariance through using voxel and SDF representations of 3D shapes. Moreover, due to the efficiency of ELM, our method runs faster than existing deep learning methods by approximately two orders of magnitude.

2.3. Extreme learning machines

Extreme learning machines (ELM) was proposed for generalized single-hidden layer feedforward neural network (SLFNs) [11,26,27] where the hidden layer need not be neuron alike. Unlike other neural networks with back propagation (BP) [28], the hidden nodes in ELM are randomly generated, as long as the activation functions of the neurons are nonlinear piecewise continuous. The weights between the hidden layer and the output layer have analytical solution and can be calculated using a formula. There are two phases in training process of ELM: feature mapping and output weights solving.

ELM feature mapping: Given input data $\mathbf{x} \in \mathbb{R}^D$, the output function of ELM for generalized SLFNs is

$$f(\mathbf{x}) = \sum_{i=1}^{L} \beta_i h_i(x) = \mathbf{h}(\mathbf{x})\beta, \tag{1}$$

where $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \cdots, h_L(\mathbf{x})]$ is the output vector of the hidden layer and $\beta = [\beta_1, \cdots, \beta_L]^T$ denotes the output weights between the hidden layer (L nodes) and the output layer (m nodes). The procedure of getting $\mathbf{h}$ is called ELM feature mapping which maps the input data from $\mathbb{R}^D$ to the feature space $\mathbb{R}^L$. In real applications, $\mathbf{h}$

[1] https://github.com/yqwang2006/CAE-ELM

can be described as

$$h_i(x) = g(\mathbf{a}_i, b_i, \mathbf{x}), \quad \mathbf{a}_i \in \mathbb{R}^D, \ b_i \in \mathbb{R}, \tag{2}$$

where $g(\mathbf{a}, b, \mathbf{x})$ is an activation function satisfying ELM universal approximation capability theorems [29]. In fact, any nonlinear piecewise continuous functions (e.g. Sigmoid, Gaussian, etc.) can be used as activation function $\mathbf{h}$. In ELM, the parameters of $\mathbf{h}$ are randomly generated based on a continuous probability distribution.

ELM output weights solving: In the second phase, given a training sample set $(\mathbf{x}_i, t_i)_{i=1}^n$ with $t_i = [0, \cdots, 0, 1_j, 0, \cdots, 0_m]^T$ the class indicator of x_i, ELM aims to minimize both the training error and the Frobenius norm of output weights. This objective function, for both binary and multi-class classification tasks, can be expressed as follows:

$$\min_{\beta,\xi} \frac{\omega}{2} \sum_{i=1}^{n} \|\xi\|_2^2 + \frac{1}{2}\|\beta\|_F^2, \quad \text{s.t. } \beta \mathbf{h}(\mathbf{x}_i) = \mathbf{t}_i - \xi_i, \quad \forall \in 1, 2, \ldots, n, \tag{3}$$

where n is the number of samples, and ξ_i denotes the training errors of the i-th sample, ω is a regularization parameter which trades off the norm of output weights and training errors, and $\|\cdot\|_F^2$ denotes the Frobenius norm.

The optimization problem in Eq. (3) can be efficiently solved. Specifically, according to the Woodbury identity [30], the optimal β can be analytically obtained as

$$\beta^\star = \begin{cases} \left(\mathbf{H}^T\mathbf{H} + \dfrac{\mathbf{I}_L}{\omega}\right)^{-1} \mathbf{H}^T\mathbf{T} & \text{if } L \le k \\ \mathbf{H}^T\left(\mathbf{H}\mathbf{H}^T + \dfrac{\mathbf{I}_n}{\omega}\right)^{-1} \mathbf{T} & \text{otherwise} \end{cases} \tag{4}$$

where $\mathbf{I}_n$ and $\mathbf{I}_L$ are identity matrices, and $\mathbf{H}$ is the hidden layer output matrix (randomized matrix) which is defined in Eq. (5):

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(x_1) \\ \vdots \\ \mathbf{h}(x_n) \end{bmatrix} = \begin{bmatrix} h_1(x_1) & \cdots & h_L(x_1) \\ \vdots & \ddots & \vdots \\ h_1(x_n) & \ldots & h_L(x_n) \end{bmatrix} \tag{5}$$

2.4. ELM variants

Though achieving great success on both theoretical and practical aspects, basic ELM can not efficiently handle large-scale learning tasks due to the limitation of memory and the intensive computational cost of the inverse of large matrices. To reduce the runtime memory, many variants, including online sequential ELM (OS-ELM) [31] and incremental ELM (I-ELM) [32], have been proposed. OS-ELM can reduce the requirement of runtime memory because the model is trained based on each chunk. I-ELM can solve memory problem by training a basic network with some hidden nodes and then adding hidden node to the existing network one by one. To relief the computational cost incurred by these operations, many variants, including partitioned-ELM [33] and parallel-ELM [34], have been proposed. Recently, a variant of OS-ELM named parallel OS-ELM (POS-ELM) [35] and parallel regularized ELM (PR-ELM) [36] are proposed to reduce training time and memory requirement simultaneously.

Hierarchical ELM variants: Recently, ELM is also extended to multi-layer structures, i.e. multi-layer ELM (ML-ELM) [23], hierarchical ELM [37], and hierarchical local receptive fields ELM (H-LRF-ELM) [38].

Similar to deep networks, ML-ELM [23] performs layer-by-layer unsupervised learning. In contrast to deep networks, ML-ELM does not require fine-tuning using back propagation (BP) which will reduce the computational cost in training process. H-ELM [37] is a new ELM-based hierarchical learning framework for multi-layer perception. H-ELM uses unsupervised multi-layer encoding for feature extraction. Unlike the greedy layer-wise training of deep learning, the layers of H-ELM are trained in a forward manner. Therefore, it has better learning efficiency than the deep learning methods. LRF-ELM was first proposed by Huang et al. [38]. In this model, the connections between the input and hidden nodes are sparse and bounded by corresponding local receptive fields (LRF). LRF-ELM learns the local structures and generates more meaningful representations at the hidden layer when dealing with image processing and similar tasks. LRF-ELM can be extended to multi-layer architecture called hierarchical LRF-ELM (H-LRF-ELM). The layers of H-LRF-ELM can be divided into two parts, namely, the feature extractor and the ELM classifier. After all, ELM-based multi-layer networks seem to provide better performance and efficiency than other deep networks.

3. Convolutional auto-encoder ELM for 3D feature learning

In this section, the model (CAE-ELM) for extracting features from 3D shapes is formulated and described in detail.

3.1. Convolutional auto-encoder ELM

CAE-ELM combines convolutional ELM and ELM AE according to the ELM learning framework, as shown in Fig. 1. This network is designed under two considerations. First, the local shared weight mechanism in convolutional networks enables our method to handle 3D shape rotation invariance properly and accelerates training time considerably. Second, the method effectively represents AE objects; thus, CAE-ELM extracts highly useful 3D features for numerous applications in computer graphics.

The CAE-ELM training procedure can be described in three phases: (1) convolutional feature map generation, where CAE-ELM generates convolutional kernels randomly and obtains convolutional feature maps with input data and random kernels. Then, the pooling operation is performed on these maps to maintain rotation invariance. (2) AE feature extraction, where we first generate random AE weights. Subsequently, the output weights of ELM AEs are computed and the input weights finally replaced with output weights. (3) ELM classifier, where all features generated by AEs are combined into a vector. This vector is expressed as the $\mathbf{H}_{final}$ matrix, and the β_{final} is computed.

The CAE-ELM structure is similar for 2D images and 3D shapes. When the input data are 2D images, CAE-ELM uses the image matrix as an input and the convolution and pooling processes as 2D operations. If the input data are 3D shapes, CAE-ELM computes the voxel and SDF data of 3D meshes and then sends these cubes to the input layer. All operations are thus performed in 3D space. The other CAE-ELM details for 2D are similar to that for 3D. For convenient presentation, we focus only on CAE-ELM implementation given 3D data.

Input representation: Given a set of 3D shapes, we generate two representation types for each shape: voxel data and SDF data. In each voxel datum, a 3D mesh is represented as a binary tensor: 1 and 0 indicate that the voxel is inside and outside the mesh surface, respectively. SDF is represented as a grid sampling of the minimum distance to the surface of an object that is represented as a polygonal model. The convention of applying negative and positive values inside and outside the object is typically applied. Moreover, SDFs are popular in computer graphics and related fields. Fig. 2 shows these two representations.

To facilitate effective feature learning, all training shapes are uniformly scaled into the unit box. Given the rotation-invariant nature of CAE-ELM, the shapes need not be oriented consistently, especially when the training set is large.

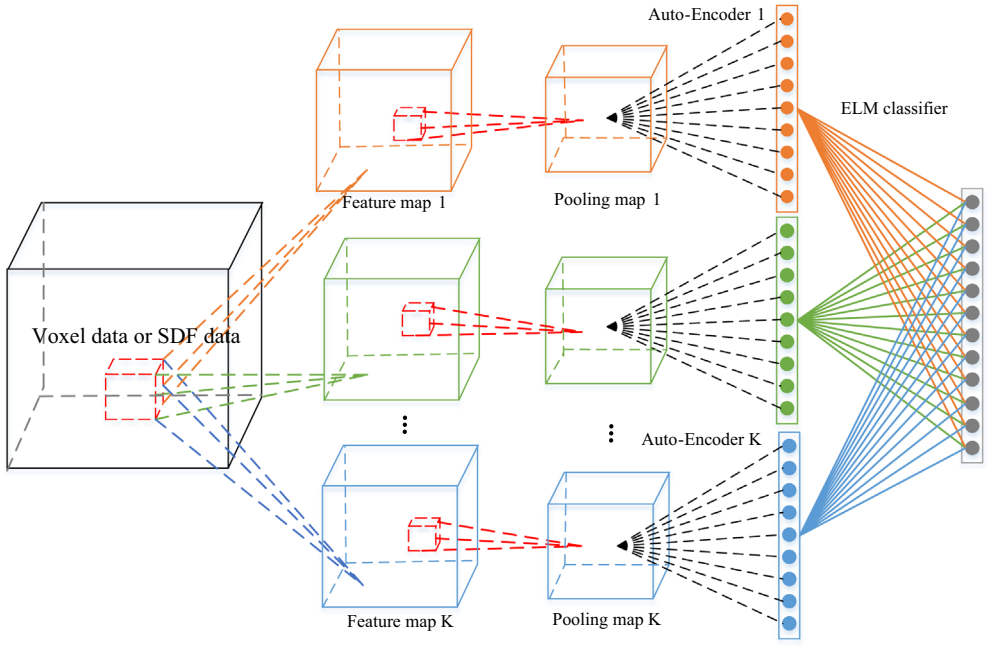

Fig. 1. The network structure of our proposed CAE-ELM.

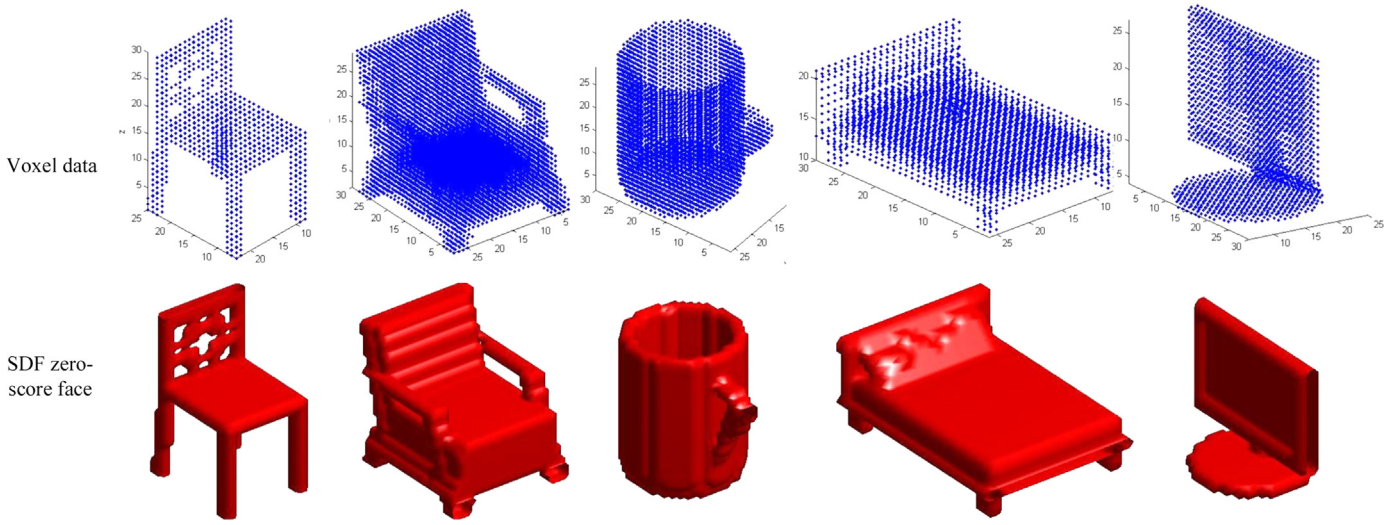

Fig. 2. The voxel points cloud and SDF zero-value iso-surface.

Convolutional feature map generation: Let us see the first phase of CAE-ELM. For each input 3D shape, we first voxelize it or compute its SDF data, and then generate k_1 feature maps (with resolution of $d \times d \times d$) produced by different randomly generated weights. Thus there are k_1 different $d \times d \times d$ feature maps in this layer. Specifically, a weight matrix $\mathbf{W}_1$ is randomly generated:

$$\mathbf{W}_1 \subset \mathbb{R}^{c_1 \times c_1 \times c_1 \times k_1}, \tag{6}$$

where the local connect size is $c_1 \times c_1 \times c_1$. The weight to the k-th feature map is denoted as

$$w^{1,k} \subset \mathbb{R}^{c_1 \times c_1 \times c_1}, \quad k = 1, \cdots, k_1. \tag{7}$$

For an input 3D shape x, the node (i,j,l) of the k-th feature map is computed as

$$l_{i,j,l,k} = \sum_{p=1}^{c_1} \sum_{q=1}^{c_1} \sum_{t=1}^{c_1} x_{i+p-1,j+q-1,l+t-1} w_{p,q,t}^{1,k}, \quad k = 1, \ldots, k_1. \tag{8}$$

The feature maps in deeper layers can be computed similarly.

Pooling: The pooling operation is designed to handle shape variance, like in CNN [39]. We choose average pooling in the network as it can well preserve the shape information.

Let us see the first layer again. Denote s_1 as the pooling size. Thus the pooling map is of size $d/s_1 \times d/s_1 \times d/s_1$. Thus the node (α, β, γ) of the k-th pooling map is computed as

$$h_{\alpha,\beta,\gamma,k} = Pooling(l_{i,j,l,k}),$$
$$i \in [\alpha - s_1, \alpha + s_1], j \in [\beta - s_1, \beta + s_1], l \in [\gamma - s_1, \gamma + s_1], \tag{9}$$

where *Pooling* is the average pooling operation.

Similarly, the pooling maps in deeper layers can be computed.

Auto-Encoder feature extraction: Given the number of feature maps after pooling is K, we need K auto-encoders to extract features, respectively. In the following, we only describe the k-th auto-encoder corresponding to the k-th feature map. Note that there is only one hidden layer in our auto-encoder layer.

Assume the number of hidden nodes in the auto-encoder is L, the output of the hidden layer of auto-encoder would be described

as

$$\mathbf{H}_k = g(\mathbf{a}_k \cdot p_k + b), \quad \mathbf{a}_k^T \mathbf{a}_k = \mathbf{I}, b_k^T b_k = 1, \tag{10}$$

where $\mathbf{a}_k = [a_{k1}, \cdots, a_{kL}]$ and $b_k = [b_{k1}, \cdots, b_{kL}]$ are the random input weights and bias between input layer and hidden layer respectively, and p_k is the k-th pooling map and the input data of the k-th auto-encoder. The output weights of the k-th auto-encoder β_k can be computed by Eq. (11):

$$\beta_k = \left(\mathbf{H}_k^T \mathbf{H}_k + \frac{1}{\omega}\mathbf{I}\right)^{-1} \mathbf{H}_k^T \mathbf{X}_k, \tag{11}$$

where $\mathbf{X}_k = [P_{k1}, \cdots, P_{kN}]$ is the output of the pooling phase, and $1/\omega$ is the normalization parameters. Then the k-th auto-encoder will get the features by using β_k and $\mathbf{X}_k$ as follows:

$$\mathbf{H}_{final\_k} = (\beta_k)^T \mathbf{X}_k, \tag{12}$$

where $\mathbf{H}_{final\_k}$ is the extracted feature by the k-th auto-encoder.

ELM classifier: The last layer of CAE-ELM is a ELM classifier. In this phase, CAE-ELM combines the output features of K auto-encoders which is shown in Eq. (13):

$$\mathbf{H}_{Total} = [\mathbf{H}_{final\_1}, \ldots, \mathbf{H}_{final\_K}] \tag{13}$$

where $\mathbf{H}_{Total}$ is the combined features. Let A_N is the number of hidden nodes of auto-encoders and N is the number of training samples. The classify layer is in full connection with the output layer, so the full connected layer matrix $\mathbf{H}_{Total} \subset \mathbb{R}^{N \times (K * A_N)}$. Then the output weights β are analytically calculated as Eq. (14):

$$\beta = \begin{cases} \left(\mathbf{H}_{Total}^T \mathbf{H}_{Total} + \frac{1}{\omega}\mathbf{I}\right)^{-1} \mathbf{H}_{Total}^T \mathbf{T}, & N \geq K * A_N, \\ \mathbf{H}_{Total}^T \left(\mathbf{H}_{Total} \mathbf{H}_{Total}^T + \frac{1}{\omega}\mathbf{I}\right)^{-1} \mathbf{T}, & \text{otherwise.} \end{cases} \tag{14}$$

where $\mathbf{T}$ is the labels of the input 3D shapes.

Testing procedure of CAE-ELM : During the testing process, the test data are subjected to the same procedure. Consequently, feature maps $\mathbf{H}$ of the testing data can be obtained. The output label value can be obtained by computing the max values of $\mathbf{H}\beta^T$.

3.2. Our architecture

We designed an architecture that can accept two types of 3D shape representation as inputs. The architecture depicted in Fig. 3 includes two parallel CAE-ELM layers to extract voxel data and SDF data features. Finally, these two features are combined into one feature vector for classification tasks. Voxel and SDF data are two types of 3D shape representation; thus, each data type has advantages and disadvantages. For example, voxel data only is expressed only as 0 and 1 in the cube when deriving the structure information of 3D shapes. In the process, many 3D shape details may be lost. By contrast, SDF data describe the SDF of 3D shapes in additional detail but with fewer global characteristics than voxel data can. Our architecture uses both voxel and SDF data as inputs; this approach adopts the advantages of voxel data while overcoming the limitations of SDF data.

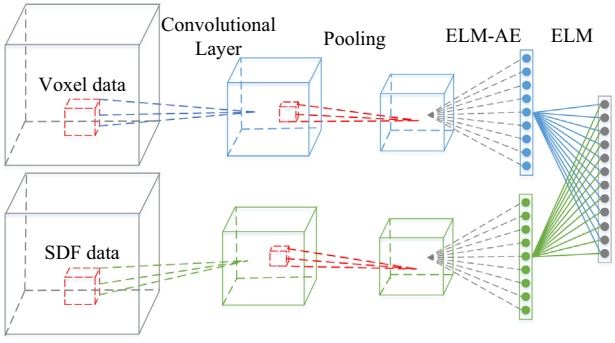

Fig. 3. The network architecture of our proposed CAE-ELM for 3D shape.

3.3. Visualization mechanism

3D shape visualization: To clarify CAE-ELM performance, we must interpret the neuron activation mechanism in the network. We visualize the features learned by CAE-ELM to enhance the interpretation. The convolutional kernels used in our networks are randomly generated; thus, we weigh K feature maps with the optimized output weights β (Eq. (14)) to produce K activation maps. These activations are then mapped back from the output layer to the original 3D mesh through inverse convolution and pooling operations that are similar to those presented in [40].

Formally, we denote H_o as the matrix of the pooling layer features of a 3D mesh. The learned feature β can be visualized through the following operations:

$$invConv(unpooling(\beta \mathbf{H}_o)), \tag{15}$$

where *invConv* and *unpooling* indicate the inverse convolution and inverse average pooling operations, respectively. Given an input 3D voxel datum x and a convolution kernel w, the convolution operation is expressed as $y = x * w$. The inverse convolution is then defined as $x' = y * flip(w)$, where *flip* corresponds to left–right and top–down flipping over the kernel along its second and third dimensions. *unpooling* is determined by returning the average value into each pixel within the pool region. As shown in Fig. 4, the activations are mapped on the input meshes through color coding. Interestingly, the maximally activated regions constitute one fixed part in one class. For example, the maximum activations of cup class occur on the bottoms of cups, and the activation of a person class is observed in the shoulders. Our CAE-ELM networks are trained discriminatively with shape classification tasks; these features essentially imply the discriminative regions of the input models in terms of shape class characterization.

2D image visualization: As mentioned previously, CAE-ELM can also accept 2D images as inputs. For convenient presentation, we visualize only the convolutional feature maps and the output AE weights. Specifically, we visualize the features of output AE weights by selecting the top m with the maximum neuron activations (output feature values). In Fig. 5, we show the feature maps and output weights of the AE phase implemented on the MNIST and Norb datasets.

4. Experiments

In this section, we demonstrate the performance of CAE-ELM and explore its applicability. First, the classification accuracy and training time of this method are determined with 3D shape datasets. Subsequently, the performance on 2D images is described. Finally, we apply the features extracted by CAE-ELM to repair broken 3D shapes. We implemented this method in MATLAB 2014b, which runs on a computer with an Intel(R) Xeon E5-2650 2.0 GHz CPU and 64 GB RAM.

4.1. 3D shape classification

Dataset: To verify the representation capability of the shape features learned by CAE-ELM, we train our network on a large collection of 3D shapes with significant intra-class variation. Princeton ModelNet is a recently released online shape dataset [10] that contains 127,915 CAD models in 662 categories; we run our algorithm on its two subsets: ModelNet10 and ModelNet40 (Table 1). The models in ModelNet10 are all manually adjusted

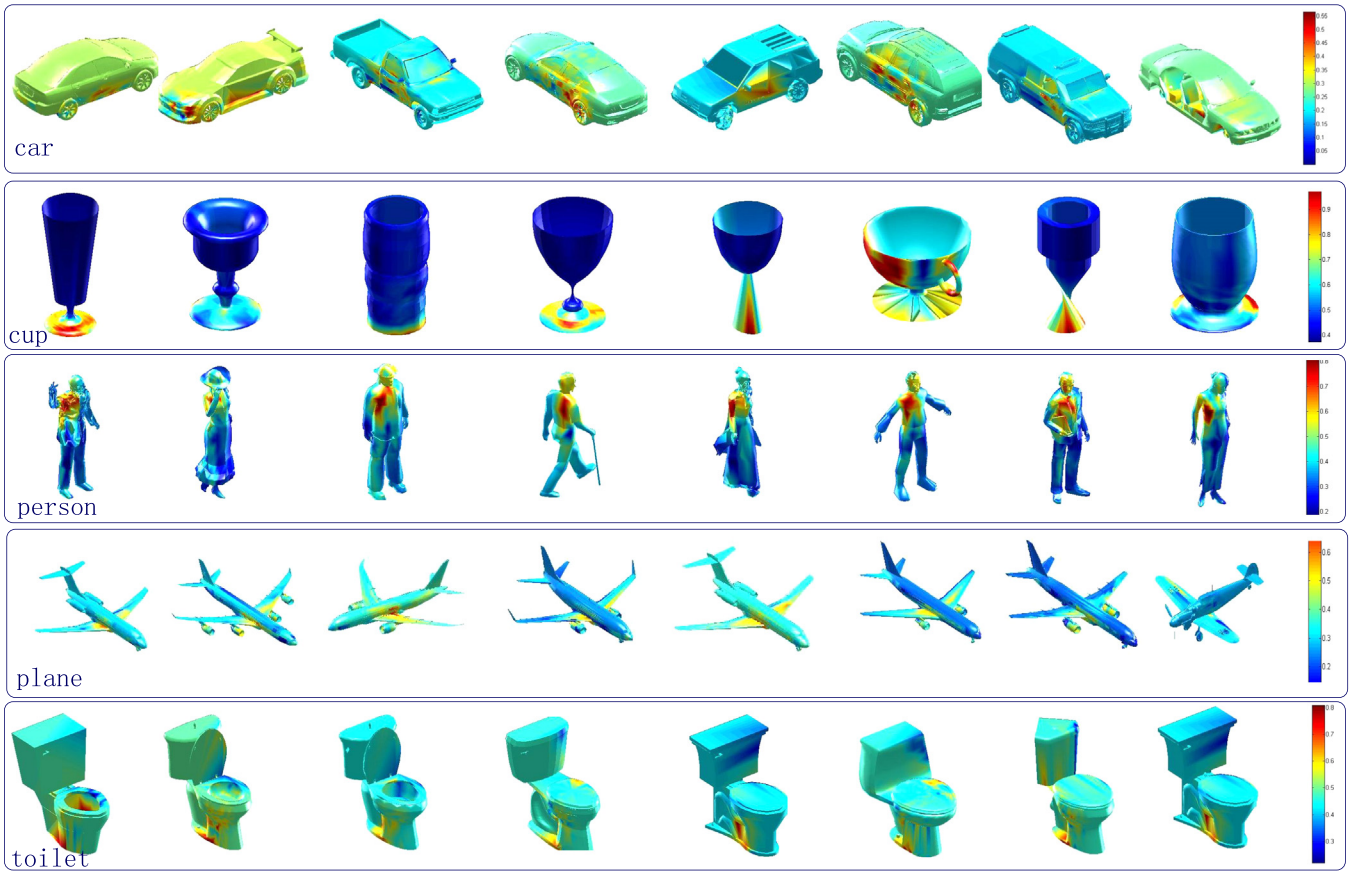

Fig. 4. Color-coded visualization of neuron activations on the input mesh.

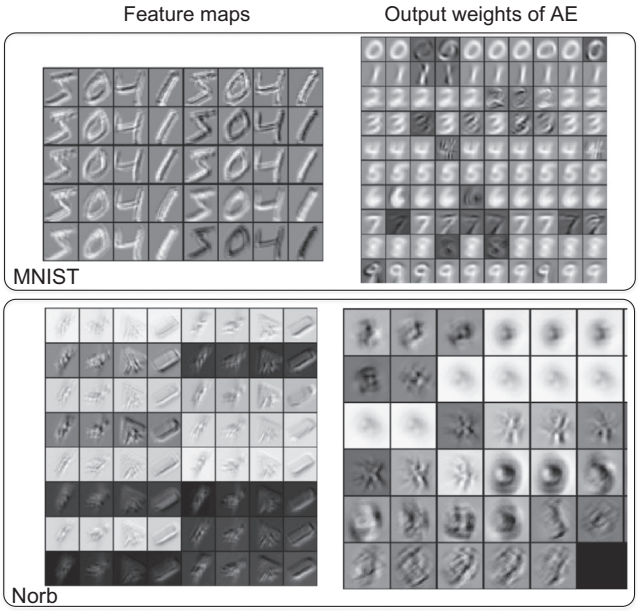

Fig. 5. Feature maps and output weights of CAE-ELM implemented on MNIST and Norb.

with consistent orientation, whereas those in ModelNet40 are not oriented. Thus, ModelNet40 can be used to test the rotation invariance of our network.

Parameter setting: We adopt the CAE-ELM network depicted in Fig. 3 in our implementation. The regularization weight $C=0.1$. Table 2 summarizes the parameters used on different datasets in our implementation.

Table 1
Statistics of ModelNet10 and ModelNet40.

Dataset	Oriented	#Models	#Training	#Testing
ModelNet10	Yes	4899	3991	908
ModelNet40	No	12,311	9843	2468

Table 2
Convolution kernel size (c), number of feature maps (K), pooling size (s), the number of hidden nodes in auto-encoder (A_N), and the normalization parameters used in auto-encoder ($1/\omega$) used by our implementation.

Dataset	K	c	s	A_N	$1/\omega$
ModelNet10	60	$5\times5\times5$	$2\times2\times2$	200	10^6
ModelNet40	100	$5\times5\times5$	$2\times2\times2$	200	10^6

Comparison with alternative classification methods: CAE-ELM consumes 250 s for training in ModelNet10 and reports 91.4% classification accuracy. By contrast, CDBN [10] takes 2 days to train a network and achieves 83.54% classification accuracy while MVD-ELM [9] requires 674 s and achieves 88.99% classification accuracy. We also contrast our learned features with two hand-crafted shape descriptors, that is, the light field descriptor (LFD) [12] and spherical harmonics (SPH) [13]. We use a linear support vector machine with the one-versus-all strategy to train classifiers and to evaluate the classification accuracy on the testing set for LFD and SPH. Table 3 concludes that CAE-ELM is more accurate than all other descriptors.

Figs. 6 and 7 display the performance of CAE-ELM on ModelNet10 and ModelNet40 when the number of convolutional

Table 3
Comparing with SVM classifier trained over handcrafted features, with CDBN with volumetric representation, and with MVD-ELM. Note that the CDBN is trained on GPU.

Dataset	Hand-crafted		Learned features				
	SPH [13]	LFD [12]	CDBN [10]	MVD-ELM [9]	CAE-ELM (voxel)	CAE-ELM (SDF)	CAE-ELM (voxel+SDF)
ModelNet10	79.97%	79.87%	83.54%	88.99%	88.66%	88.77%	91.41%
	–	–	2 days	674 s	151.75 s	317.6 s	250 s
ModelNet40	68.23%	75.47%	77.32%	81.39%	82.66%	80.11%	84.35%
	–	–	>2 days	306.4 s	609 s	610 s	1224 s

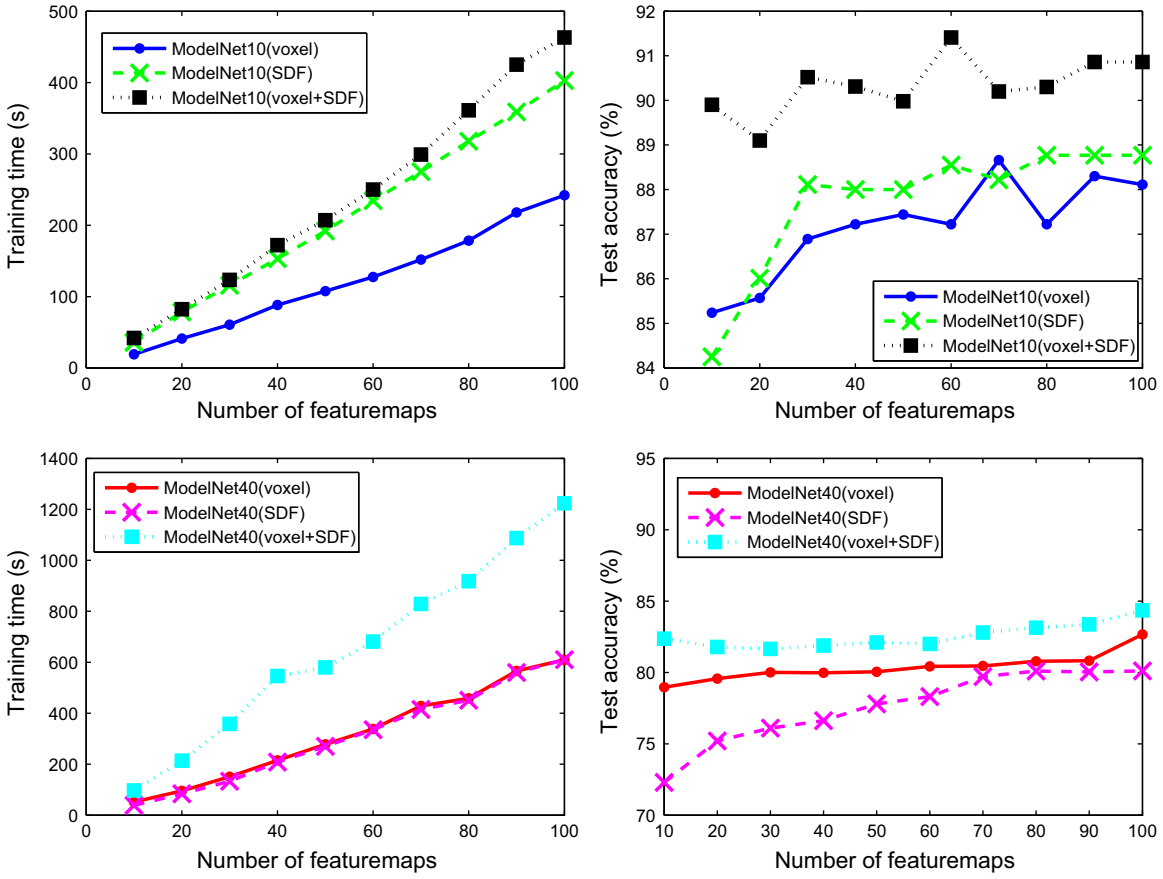

Fig. 6. Performance of CAE-ELM on ModelNet10 and ModelNet40 with varying the number of convolutional kernels. The hidden nodes' number of auto-encoder is 216.

kernels or the number of hidden nodes of auto-encoder is varied. It can be concluded from Fig. 6 that the classification accuracy of the CAE-ELM that accepts two types of 3D shape representation is higher than that of the CAE-ELM that accepts only a single type of representation. It can be seen from Fig. 7 that the accuracy curves remain stable at around 91% (ModelNet10) or 82%(ModelNet40) when the number of hidden nodes of auto-encoder is increasing.

Rotation invariance of CAE-ELM : To verify the rotation invariance of CAE-ELM, we run this method on ModelNet40, where the models are not oriented. As with [10], we rotate each model by 30 degree to generate additional model instances in arbitrary poses. The training and testing sets are arranged to avoid overlapping models, even for the same model with different poses. This experiment is executed on a workstation with 128 GB RAM. As indicated in Table 3, our method requires 1224 s for training and reports 84.35% classification accuracy, whereas MVD-ELM [9], CDBN [10], LFD [12], and SPH [13] achieve 81.39%, 77.32%, 75.47%, and 63.59% accuracy, respectively.

3D shape retrieval: We apply the L2 norm to measure the similarity in the shapes of each pair of testing samples for 3D retrieval, as in [10]. To evaluate our retrieval algorithms, we exhibit the precision–recall curves of CAE-ELM and of three other corresponding methods in Fig. 8. CAE-ELM has higher precision and recall than the other methods, thus indicating that CAE-ELM also reports superior retrieval performance.

4.2. CAE-ELM on 2D image

Dataset: To verify the 2D representation capability of the features learned by CAE-ELM, we run our algorithm on two classical image datasets, namely, MNIST and Norb. MNIST is a handwriting number dataset with 60,000 training samples and 10,000 testing instances, whereas Norb has 24,300 training sample pairs and 24,300 testing sample pairs. Each image pair generates two projections of one 3D shape.

Parameter setting: We adopt a single-layer network in 2D CAE-ELM implementation. Table 4 summarizes the parameters used on different datasets in our implementation.

Comparison with deep learning classification methods: Our 2D CAE-ELM requires 1090 s for training and reports 98.87%

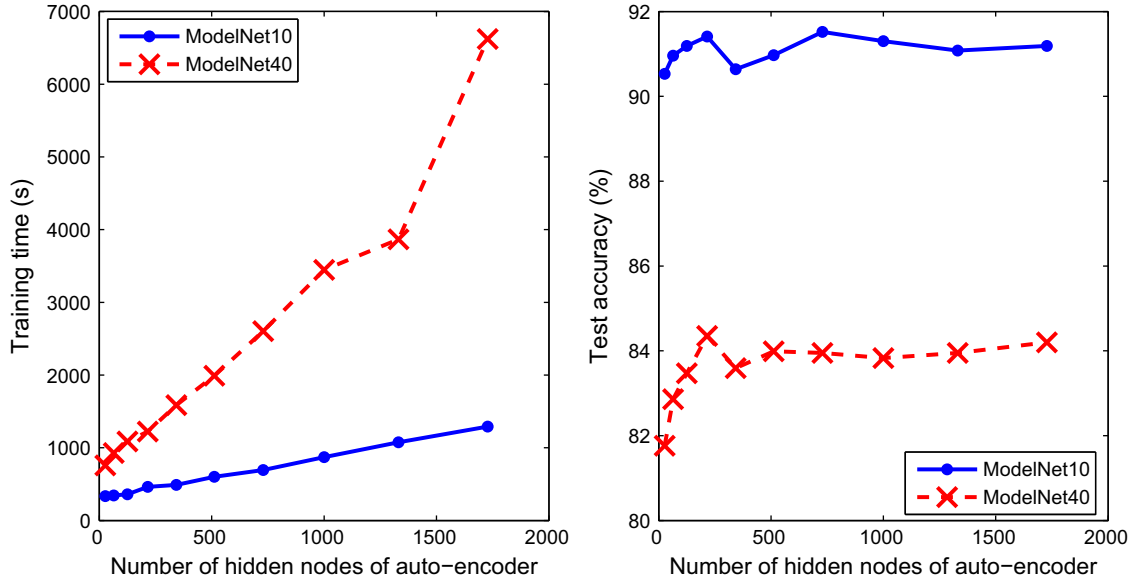

Fig. 7. Performance of CAE-ELM on ModelNet10 and ModelNet40 with varying the number of hidden nodes of auto-encoder. The number of feature maps is 100.

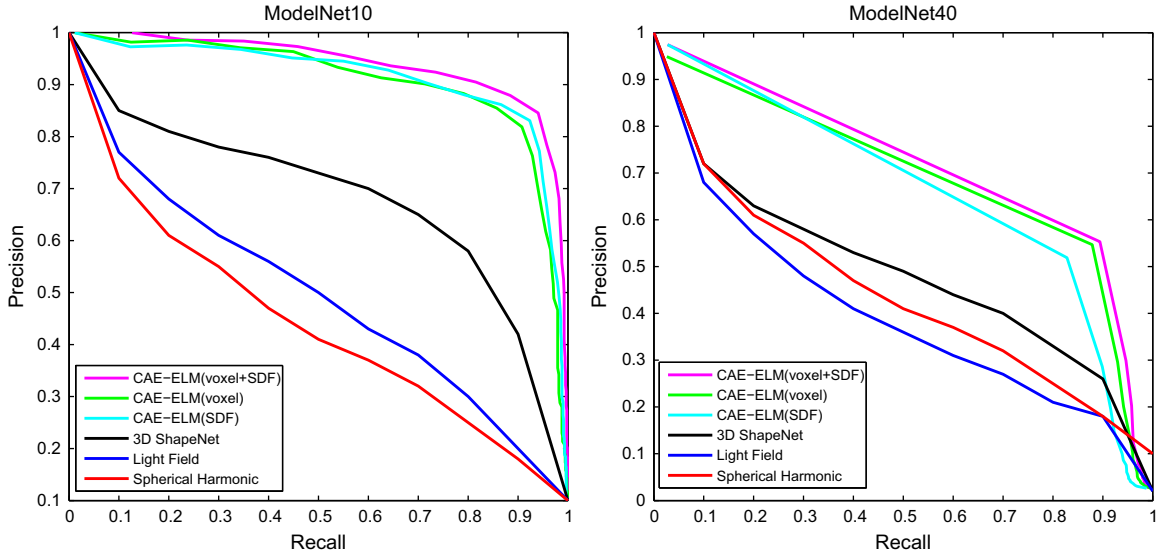

Fig. 8. 3D Mesh Retrieval. Precision–recall curves of SPH, LFD, ShapeNets, and CAE-ELM.

Table 4
Normalized 2D convolution kernel size (c), number of feature maps (K), pooling size (s), the number of hidden nodes in auto-encoder (A_N), and the normalization parameters used in auto-encoder ($1/\omega$) used by our implementation.

Dataset	K	c	s	A_N	$1/\omega$
MNIST	100	5×5	2×2	144	10^6
Norb	80	5×5	2×2	196	10^6

Table 5
The performance of 2D CAE-ELM compared with deep learning methods.

Dataset	DBN	SAE	SDAE	CAE-ELM
MNIST	98.87% 5.7 h	98.6% 19 h	98.7% 17 h	98.87% 1090 s
Norb	92.8% 15104 s	93.5% 85717 s	94.4% 53378 s	94.5% 1208 s

classification accuracy on the MNIST dataset. By contrast, the deep belief network, stacked AE, and stacked denoising AE algorithms take 5.7, 19, and 17 h to train their networks and achieve testing accuracies of 98.87%, 98.6%, and 98.7%, respectively. Table 5 shows that CAE-ELM has the shortest training time of all the methods at the same classification accuracy.

Number of convolutional kernels: The output accuracy of CAE-ELM is expected to improve with an increase in the number of feature maps. To demonstrate the effect of view count, we run CAE-ELM on the MNIST and Norb datasets with different numbers of feature maps. As shown in Fig. 9, the classification accuracy of CAE-ELM increases with the number of feature maps.

4.3. 3D Shape completion

In this section, we introduce the 3D shape completion procedure with CAE-ELM. First, we train a CAE-ELM network using complete shapes and store the features before the classifier, that is, H_{Total}. Then, we utilize broken shapes as the CAE-ELM input and obtain the H_{broken}. Subsequently, we compare H_{broken} with all H_{Total} of training samples

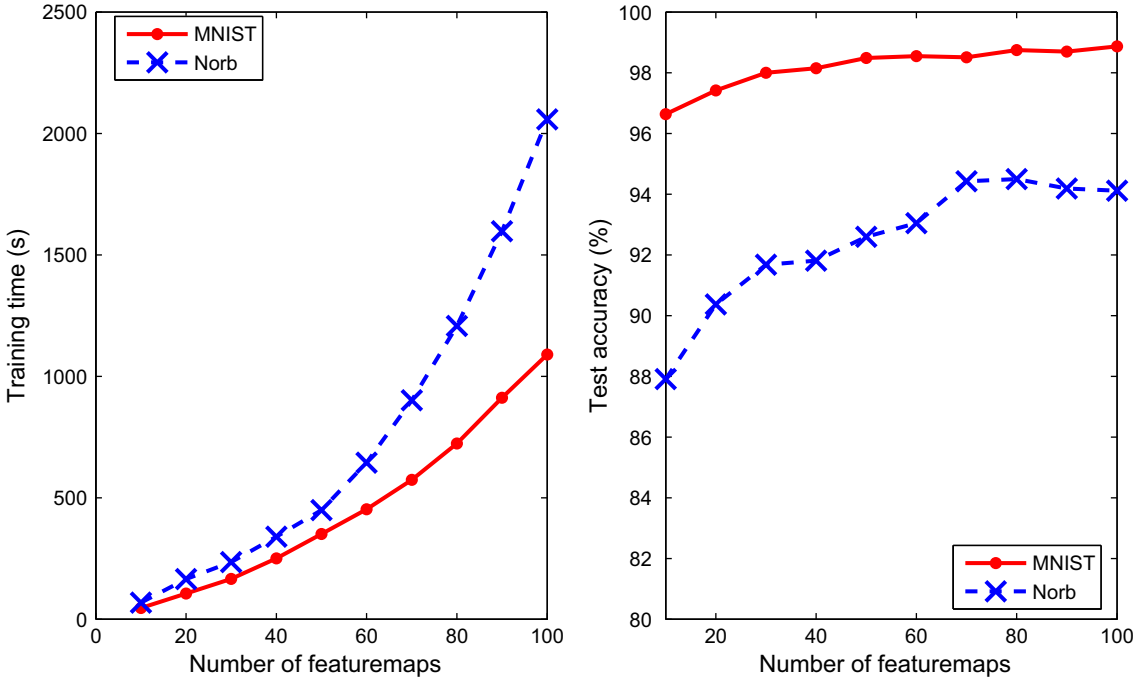

Fig. 9. Performance of CAE-ELM on MNIST and Norb with varying number of convolutional kernels.

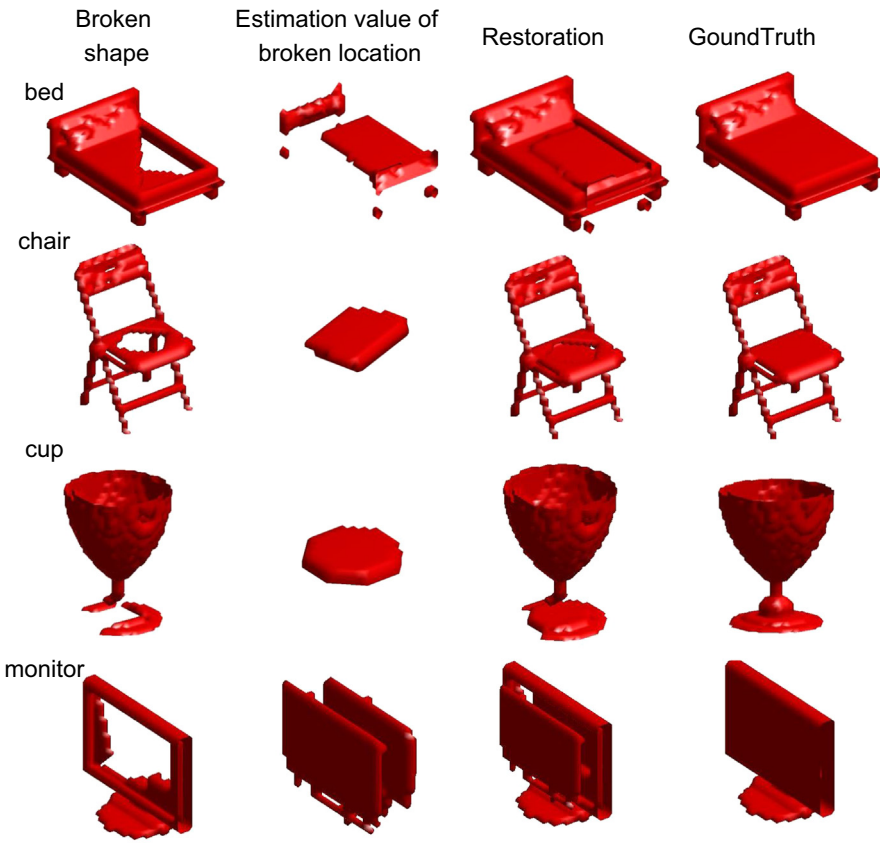

Fig. 10. Incomplete 3D shapes and their completion using CAE-ELM. The columns from left to right show estimation of hole regions, shape completion results and groud-truth data.

and then obtain the K (in this experiment, $K=10$) along with the nearest features of H_{broken} in H_{Total}. We compute the average vector of the K features known as $H_{average}$ and obtain $delta_H$. The difference between the result of H_{broken} and $H_{average}$ is calculated. Furthermore, we believe that the locations that must be repaired in 3D shapes differ in value from those in the complete shapes; therefore, $delta_H$ can be used for completion. Finally, we use $delta_H$ to reconstruct the input data and to obtain the delta shape. Subsequently, we patch the delta shape onto the broken shape and generate a repaired shape. Fig. 10 shows the results of our 3D shape completion procedure.

5. Conclusion

In this paper, we propose a new method called CAE-ELM that can extract features from 3D shapes. In contrast to existing 3D shape feature learning methods, our method combines the advantages of convolution, pooling, and AE processes; moreover, this technique uses both voxel and SDF data as inputs to improve performance. In the future, we will examine this approach further in three directions. First, the CAE-ELM in this work is a single-layer network. Multi-layer models can extract considerably more high-level features than single-layer models can; thus, our model should be developed as a multi-layer model. Second, CAE-ELM can be applied in other fields; some possible applications include 3D object recognition in cluttered indoor scenes, 3D scene understanding for robot navigation, and 3D object analysis for robot manipulation. In the robotics setting, online feature learning can be developed such that the training data may originate from robot actions. Finally, new methods can be developed for learning comprehensive 3D features that fully cover the geometry and structure of input shapes.

Acknowledgements

The authors would like to thank the anonymous reviewers for their helpful comments. This work was supported by the National Natural Science Foundation of China (No. 61125201, 61402499, 61379103 and U1435219).

References

[1] F.-w. Qin, L.-y. Li, S.-m. Gao, X.-l. Yang, X. Chen, A deep learning approach to the classification of 3d cad models, J. Zhejiang University SCIENCE C 15 (2) (2014) 91–106.
[2] S. Bu, P. Han, Z. Liu, J. Han, H. Lin, Local deep feature learning framework for 3d shape, Comput. Graph. 46 (2015) 117–129.
[3] Z. Wu, S. Song, A. Khosla, X. Tang, J. Xiao, 3d shapenets for 2.5d object recognition and next-best-view prediction, ArXiv e-prints http://arXiv:1406.5670arXiv:1406.5670 (June 2014).
[4] Z. Zhu, X. Wang, S. Bai, C. Yao, X. Bai, Deep learning representation using autoencoder for 3d shape retrieval, CoRR abs/1409.7164.
[5] B. Leng, X. Zhang, M. Yao, Z. Xiong, A 3d model recognition mechanism based on deep Boltzmann machines, Neurocomputing 151, Part 2 (0) (2015) 593–602.
[6] Y. LeCun, Y. Bengio, Convolutional networks for images, speech, and time series, The handbook of brain theory and neural networks 3361 (10) 1995.
[7] B. Leng, S. Guo, X. Zhang, Z. Xiong, 3d object retrieval with stacked local convolutional autoencoder, Signal Process. 112 (0) (2015) 119–128.
[8] Z. Zhu, X. Wang, S. Bai, C. Yao, X. Bai, Deep learning representation using autoencoder for 3d shape retrieval, arXiv preprint http://arXiv:arXiv:1409.7164arXiv:arXiv:1409.7164.
[9] Z. Xie, K. Xu, W. Shan, L. Liu, Y. Xiong, H. Huang, Projective feature learning for 3d shapes with multi-view depth images, in: Proceedings of Pacific Graphics, 2015.
[10] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, J. Xiao, 3d shapenets: a deep representation for volumetric shapes, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1912–1920.
[11] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1) (2006) 489–501.
[12] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, M. Ouhyoung, On visual similarity based 3d model retrieval, in: Computer graphics forum, vol. 22, Wiley Online Library, 2003, pp. 223–232.
[13] M. Kazhdan, T. Funkhouser, S. Rusinkiewicz, Rotation invariant spherical harmonic representation of 3d shape descriptors, in: Symposium on Geometry Processing, vol. 6, 2003, pp. 156–164.
[14] T. Funkhouser, P. Min, M. Kazhdan, J. Chen, A. Halderman, D. Dobkin, D. Jacobs, A search engine for 3d models, ACM Trans. Graph. (TOG) 22 (1) (2003) 83–105.
[15] L. Shapira, S. Shalom, A. Shamir, D. Cohen-Or, H. Zhang, Contextual part analogies in 3D objects, Int. J. Comput. Vis. 89 (2–3) (2010) 309–326.
[16] P. Heider, A. Pierre-Pierre, R. Li, C. Grimm, Local shape descriptors, a survey and evaluation, in: Proceedings of Eurographics Workshop on 3D Object Retrieval, 2011, pp. 49–56.
[17] A.M. Bronstein, M.M. Bronstein, L.J. Guibas, M. Ovsjanikov, Shape google: geometric words and expressions for invariant shape retrieval, ACM Trans. Graph. (TOG) 30 (1) (2011) 1.
[18] S. Belongie, J. Malik, J. Puzicha, Shape matching and object recognition using shape contexts, IEEE Trans. Pattern Anal. Mach. Intell. 24 (4) (2002) 509–522.
[19] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, M. Ouhyoung, On visual similarity based 3d model retrieval, Comput. Graph. Forum 22 (3) (2003) 223–232.
[20] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[21] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[22] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neural Comput. 18 (7) (2006) 1527-1554.
[23] L.L.C. Kasun, H. Zhou, G.-B. Huang, C.M. Vong, Representational learning with elms for big data, IEEE Intell. Syst. 28 (6) (2013) 31–34.
[24] Y. Bengio, A.C. Courville, P. Vincent, Unsupervised feature learning and deep learning: a review and new perspectives, CoRR abs/1206.5538. URL ⟨http://arxiv.org/abs/1206.5538⟩.
[25] X. Zhang, X. Le, A. Panotopoulou, E. Whiting, C.C. Wang, Perceptual models of preference in 3d printing direction, ACM Trans. Graph. (Proc. SIGGRAPH), 2015.
[26] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16) (2007) 3056–3062.
[27] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Trans. Syst. Man Cybern. Part B: Cybern. 42 (2) (2012) 513–529.
[28] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning representations by back-propagating errors, Cognit. Model. 5 (1988) 3.
[29] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (4) (2006) 879–892.
[30] M.A. Woodbury, Inverting modified matrices, Memo. Rep. 42 (1950) 106.
[31] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, IEEE Trans. Neural Netw. 17 (6) (2006) 1411–1423.
[32] G.-B. Huang, L. Chen, Convex incremental extreme learning machine, Neurocomputing 70 (16) (2007) 3056–3062.
[33] J. Lin, J. Yin, Z. Cai, Q. Liu, K. Li, V. Leung, A secure and practical mechanism of outsourcing extreme learning machine in cloud computing, IEEE Intell. Syst. 28 (6) (2013) 35–38.
[34] Q. He, T. Shang, F. Zhuang, Z. Shi, Parallel extreme learning machine for regression based on mapreduce, Neurocomputing 102 (2013) 52–58.
[35] B. Wang, S. Huang, J. Qiu, Y. Liu, G. Wang, Parallel online sequential extreme learning machine based on mapreduce, Neurocomputing 149 (2015) 224–232.
[36] Y. Wang, Y. Dou, X. Liu, Y. Lei, Pr-elm: parallel regularized extreme learning machine based on cluster, Neurocomputing, http://dx.doi.org/10.1016/j.neucom.2015.08.066.
[37] J. Tang, C. Deng, G.-B. Huang, Extreme learning machine for multilayer perceptron. IEEE Trans. on Neural Netw. and Learning Syst. 2015, http://dx.doi.org/10.1109/TNNLS.2015.2424995.
[38] G.-B. Huang, Zuo Bai, C.M.V. Liyanaarachchi Lekamalage Chamara Kasun, C.L.P. Chen, Local receptive fields based extreme learning machine, IEEE Comput. Intell. Mag. 10 (2), 2015.
[39] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. 2012, 1097-1105.
[40] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional neural networks, arXiv preprint arXiv:1311.2901.

Yueqing Wang was born in 1988. He received his B.S. degree in Computer Science and Technology in Tsinghua University, in 2010, and received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2012, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include high performance computer architecture, parallel computing, and machine learning.

Zhige Xie was born in 1984. He received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2010, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include computer graphics and machine learning.

Kai Xu was born in 1982. He is an Assistant Professor at School of Computer Science, National University of Defense Technology (NUDT). His research interests are in the areas of computer graphics, especially in geometry processing. His current topics of interests include shape analysis, high-level geometry processing, and data-driven shape modeling.

Yong Dou was born in 1966, professor, Ph.D. supervisor, senior membership of China Computer Federation (E200009248). He received his B.S., M.S., and Ph.D. degrees in Computer Science and Technology at National University of Defense Technology in 1995. His research interests include high performance computer architecture, high performance embedded microprocessor, reconfigurable computing, and bioinformatics. He is a member of the IEEE and the ACM.

Yuanwu Lei was born in 1982. He received his B.S. degree in Computer Science and Technology in North China Electric Power University, Baoding, China, in 2005, and received his M.S. degree in Computer Science and Technology at National University of Defense Technology in 2007, and now he is a Ph.D. candidate at National University of Defense Technology. His research interests include high performance computer architecture, high-precision computation, parallel computing, and reconfigurable computing.