Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A deep learning approach to the classification of 3D CAD models*

Fei-wei QIN, Lu-ye LI, Shu-ming GAO‡, Xiao-ling YANG, Xiang CHEN

(*State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310058, China*)

E-mail: qinfeiwei@zjucadcg.cn; {liluye, smgao, xchen}@cad.zju.edu.cn; sunny_aday@163.com

Received July 9, 2013; Revision accepted Nov. 21, 2013; Crosschecked Jan. 15, 2014

Abstract: Model classification is essential to the management and reuse of 3D CAD models. Manual model classification is laborious and error prone. At the same time, the automatic classification methods are scarce due to the intrinsic complexity of 3D CAD models. In this paper, we propose an automatic 3D CAD model classification approach based on deep neural networks. According to prior knowledge of the CAD domain, features are selected and extracted from 3D CAD models first, and then pre-processed as high dimensional input vectors for category recognition. By analogy with the thinking process of engineers, a deep neural network classifier for 3D CAD models is constructed with the aid of deep learning techniques. To obtain an optimal solution, multiple strategies are appropriately chosen and applied in the training phase, which makes our classifier achieve better performance. We demonstrate the efficiency and effectiveness of our approach through experiments on 3D CAD model datasets.

Key words: CAD model classification, Design reuse, Machine learning, Neural network
doi:10.1631/jzus.C1300185 **Document code:** A **CLC number:** TP391.72

1 Introduction

Designers spend about 60% of their time searching for the right information during the product design process and 80% of their design could be created from an existing CAD model or by modifying an existing CAD model (Gunn, 1982). This shows that the retrieval and reuse of CAD models are very important. However, massive and complex CAD models generated by previous product development activities are usually disorderly archived in enterprises, which makes design reuse a difficult task (Bai *et al.*, 2010).

CAD model classification plays a key role in effective management and organization of a large number of CAD models. Traditionally, 3D CAD model classification is primarily achieved by a time consuming and troublesome manual process, during which errors often come up. Therefore, an automatic and intelligent classification approach is of significance. Due to the intrinsic sophistication of 3D model classification problems (for example, the features and parameters of models may vary significantly depending on product families), no rigid rules, which are robust and general enough, could be applied in model category recognition. In previous works, scholars usually tended to conquer the 3D model classification problem by exploiting machine learning techniques. However, constrained by the past development of machine learning techniques (for example, extracting distinctive features from raw 3D data is difficult, and the widely used support vector machine (SVM) classifier has relatively few parameters to tune), these approaches are not mature enough to be used in industrial production.

In this paper, we propose a deep learning approach to automatically classify 3D CAD models according to the mechanical part catalogue. The designed deep neural network classifier is based on the latest machine learning technique, deep learning,

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. 61163016 and 61173125)

which is closer to the cognitive custom of engineers when they are conducting CAD model classification (Bengio, 2009; Bengio *et al.*, 2013). To the best of our knowledge, no past research exists on how to use deep learning techniques to train classification systems for 3D CAD models.

The contributions of this paper are summarized as follows:

1. We have successfully applied deep learning, a breakthrough technology in the machine learning research community, to automatic classification of CAD models for the first time.

2. According to characteristics of the CAD domain, we choose and extract corresponding key features from 3D CAD models and preprocess them as input vectors for category recognition.

3. Analyzing the thinking process of how engineers recognize 3D CAD models, we construct an automatic classifier based on deep neural networks. Also, several training strategies are properly chosen to find the optimal solution of hyper parameters in the constructed networks, which makes the classifier achieve better performance.

2 Related works

This research aims to classify 3D CAD models into corresponding semantic categories. Our work has been inspired by recent progress in several different areas such as 3D model classification and deep learning.

2.1 3D model classification

Due to its wide applications in a variety of areas, 3D model classification has drawn much attention in recent years. Limited by the representation abilities of 3D shape descriptors and the development of machine learning technologies, however, the research results are not very satisfactory.

Some scholars aim mainly at solving engineering CAD model classification problems. Wu and Jen (1996) presented a neural network approach to the classification of 3D prismatic parts. In this approach, a 3D part was modeled by the contours of its three projected views, and then converted to input vectors for a polygon classifier. In the experiment, 36 workpieces were classified in this way. However, this approach considers only the contour information of the 3D workpiece projections. The category distinguishing ability of this approach is not good enough. Ip *et al.* (2003) and Ip and Regli (2005a) presented a machine learning approach to the classification of mechanical CAD models. The enhanced shape distribution was used to convert mesh representation of CAD models into histograms, and then the k nearest neighbor (kNN) algorithm was chosen to classify solid models. Parameters of the algorithm were tuned during the training phase. The performance of the kNN classifier was illustrated through experiments, but the success rates were not satisfactory. Moreover, Ip and Regli (2005b) used SVM to classify prismatic machined and cast-then-machined parts. Four kinds of surface curvatures were computed as input vectors for the SVM based classifier. Experimental results showed that the minimum curvature acquired the highest classification accuracy. Hou *et al.* (2005) presented an SVM based clustering approach to organize 3D CAD models semantically. In this approach, each input vector is a hybrid representation of three kinds of features, including moment invariants, geometric ratios, and principal moments, through which the CAD model is represented from different perspectives. During experiments 218 3D CAD models belonging to six part families were tested, and the overall error was 11.76%.

In addition, for general 3D shape classification, other scholars proposed some approaches to deal with corresponding situations. Barutcuoglu and DeCoro (2006) presented a hierarchical shape classification approach based on a Bayesian framework. Given a set of independent classifiers for an arbitrary type of input vector, the Bayesian aggregation algorithm uses the results of these classifiers, resolves their probable inconsistency, and produces more accurate predictions. Experiments on the Princeton Shape Benchmark showed that the proposed approach improves the classification accuracy of the majority of classes. Wei *et al.* (2008) presented a Hopfield neural network approach to classify 3D VRML models considering their material colors. The appearance colors were used as input vectors for the Hopfield neural network classifier. Experiments on 30 VRML models showed the effectiveness of the proposed approach. Wang *et al.* (2013) proposed a new mechanism which can automatically select the appropriate descriptors to

retrieve and classify 3D models. First, several histogram based shape descriptors were calculated to form the feature space; second, important descriptors were selected automatically with the sparse theory; finally, a new shape descriptor was obtained using spectral clustering. The proposed mechanism worked well for both complete and incomplete models. However, the descriptors selected must be histogram based ones.

2.2 Deep learning

Deep learning makes machine learning take a big step toward its original goal, i.e., implementing true artificial intelligence. The deep learning approach has obtained significant breakthroughs since 2006, and also triggered a research boom in machine learning and artificial intelligence communities. It is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text (Bengio, 2009). Motivations for deep architectures are the following: insufficient depth can hurt; the brain has a deep architecture; and cognitive processes seem deep. Currently, no application of the deep learning approach exists in CAD/CAM domains, though deep learning has been comprehensively applied to data reduction, object recognition, image classification, etc.

Hinton and Salakhutdinov (2006) proposed a new data dimensionality reduction approach with deep neural networks. High-dimensional data can be converted to low-dimensional codes by training a multilayer neural network with a small central layer to reconstruct high-dimensional input vectors. Hidden layers of the networks are first pre-trained in an unsupervised way, and then gradient descent is used to fine-tune the weights in the neural networks. The proposed approach has better performance than the principal component analysis (PCA) based dimensionality reduction approaches. Huang and LeCun (2006) presented a hybrid architecture which combines convolutional neural network (CNN) and SVM to recognize generic objects. CNN is used to learn features from the original data first, and then the learned features are used as input vectors for training a Gaussian-kernel SVM. Experiments on the NORB datasets showed that the hybrid architecture obtains lower error rates than using CNN or SVM alone. Kavukcuoglu *et al.* (2010) proposed an approach of visual feature learning in an unsupervised way. Through convolutional training and redundancy reduction, multi-stage hierarchies of sparse convolutional features could be learned for visual recognition and detection. Krizhevsky *et al.* (2012) trained a deep convolutional neural network to classify more than 1 million images into 1000 different classes. The deep neural network which consists of five convolutional layers, three fully connected layers, and a final 1000-way softmax is very large, containing more than 650 000 neurons and 60 million trainable parameters. It achieved the lowest error rates in the ImageNet LSVRC-2010 contest.

3 Overview of our automatic 3D CAD model classification approach

Three-dimensional CAD model classification is a highly intelligent activity. The engineers who classify the models manually should have rich knowledge and experience in this domain, and need to go through some complex thinking processes to accomplish this work. The cognitive processes of the engineers seem deep: Engineers organize their ideas and concepts hierarchically; engineers learn simpler concepts and compose them to represent more abstract ones; engineers break up solutions into multiple levels of abstraction and processing.

Since deep learning can well simulate the thinking process of the human brain, we propose an automatic 3D CAD model classification approach with the aid of deep learning. The core of this approach is designing a deep neural network classifier for 3D CAD models. Fig. 1 shows the overall workflow of our approach.

The pipeline of our approach mainly consists of six steps:

1. Acquire enough sample data from real manufacturing enterprises, and build 3D CAD model data sets for training, validation, and testing.

2. Analyze the representation ability of commonly used 3D shape descriptors, select one or a group of them, and then extract features from CAD models.

3. Preprocess the input pattern and generate an input vector for the classifier.

4. Construct the deep neural network as the 3D CAD model classifier, including designing the topology of the deep architecture, allocating a certain number of neurons to every hidden layer, and so on.

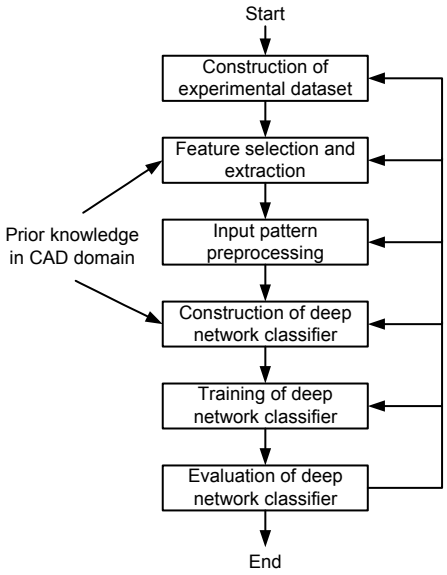

Fig. 1 Overall workflow of our approach

5. Train the 3D CAD model classifier and obtain the optimal solution.

6. Evaluate the acquired classifier and apply it to unknown 3D CAD model databases. If the classification accuracy is low, go back to the previous steps to retrain the classifier. Such a process is iterated until the required classification accuracy is reached.

4 Generation of the input vector

4.1 Experimental dataset

Until now, there has been no generally accepted benchmark for 3D CAD models. In this study, we build our own 3D CAD model database as the test bed for the research. All the CAD models are collected from several mechanical manufacturing enterprises. The models are designed by experienced engineers with mainstream commercial CAD toolkits such as SolidWorks, Pro/Engineer, CATIA, and UG NX. There are totally 7464 models belonging to 28 generic categories: gears, screws, nuts, springs, wheels, keys, bearing houses, flanges, washers, etc. The mechanical part catalog is used as a reference for selecting those categories. The whole model dataset is divided into 5990 samples for training, 737 samples for validation, and 737 samples for testing. The training set and validation set are used to perform model selection and hyper parameter selection, whereas the test set is used to evaluate the final generalization error and compare different classifiers in an unbiased way. Fig. 2 shows a portion of the 3D models in the dataset.

4.2 Feature selection and extraction

According to the specific nature of the problem domain, selecting features that have obvious distinguishable meaning is a critical step in the pipeline of our approach. The features should be invariant to irrelevant deformation, insensitive to noise, and very effective for distinguishing different categories of CAD models.

There has been much research on extracting features from 3D models, most aiming to propose powerful descriptors for representing raw 3D data. The existing 3D shape descriptors are broadly classified into three categories: statistic-based, topology-based, and view-based (Iyer *et al.*, 2005; Bimbo and Pala, 2006).

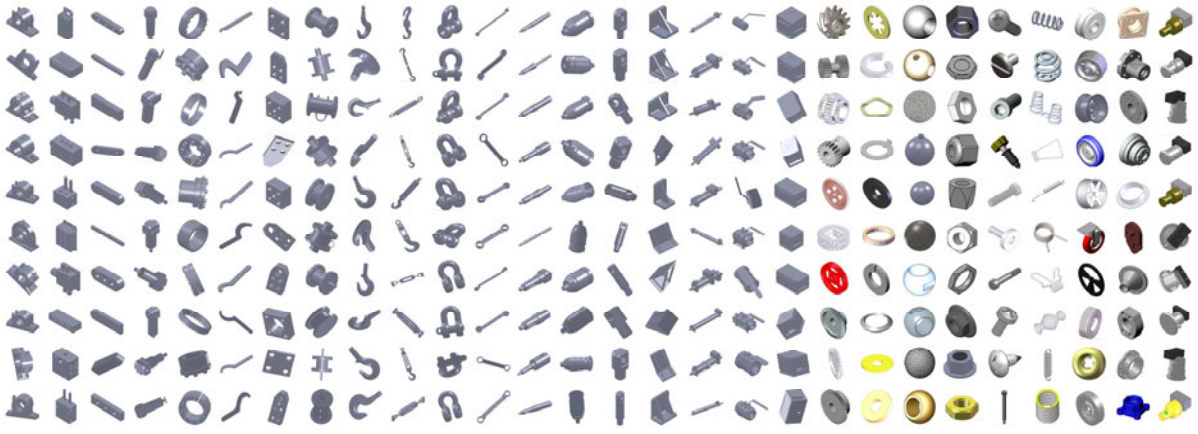

Fig. 2 Some 3D CAD models in the dataset

Each column represents a model class, and the 10 models displayed in each column are randomly selected from the corresponding model categories

In our opinion, view-based descriptors such as the light field descriptor (LFD) (Chen, 2003; Chen *et al.*, 2003) are closer to human perception as compared to the other two categories. This is because engineers usually open a 3D model with CAD software, rotate it on the screen, observe it carefully from desirable viewing angles, and then synthesize those viewing images to obtain the semantic categories. Two 3D models belonging to the same mechanical part catalogue look similar from all viewing angles. As LFD performs best according to the test carried out by Shilane *et al.* (2004), we employ it to characterize the shape information of the 3D model.

Using LFD, features of the images need to be extracted further after 2D images are generated from 3D models through light field projection. Candidate image descriptors include region-based descriptors (e.g., the Zernike moments descriptor) and contour-based descriptors (e.g., the Fourier descriptor) (Zhang and Lu, 2002). The Zernike moments descriptor is derived from complex Zernike polynomials over the unit circle ($x^2+y^2\leq 1$):

$$V_{nm}(\rho,\theta)=R_{nm}(\rho)\exp(\mathrm{j}m\theta), \tag{1}$$

$$R_{nm}(\rho)=\sum_{s=0}^{(n-|m|)/2}(-1)^s\frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!}\rho^{n-2s}, \tag{2}$$

where n is the order and m is the repetition, satisfying $n-|m|$=even and $|m|\leq n$. The Zernike moments of order n with repetition m are expressed as

$$A_{nm}=\frac{n+1}{\pi}\sum_x\sum_y f(x,y)V_{nm}(\rho,\theta). \tag{3}$$

The Zernike moments descriptor takes into account all the pixels within a shape region. The Fourier descriptor is obtained through Fourier transform on a shape signature function derived from boundary coordinates $\{(x_i, y_i), i=1, 2, \ldots, N\}$. A commonly used shape signature function is the centroid distance function which is given by the distance from the boundary points to the centroid of the shape:

$$r_i=\sqrt{(x_i-x_\mathrm{c})^2+(y_i-y_\mathrm{c})^2},\quad i=1,2,...,N, \tag{4}$$

where

$$x_\mathrm{c}=\frac{1}{N}\sum_{i=1}^{N}x_i,\quad y_\mathrm{c}=\frac{1}{N}\sum_{i=1}^{N}y_i. \tag{5}$$

The discrete Fourier transform is then applied on r_i to obtain the coefficients:

$$a_n=\frac{1}{N}\sum_{i=1}^{N}r_i\exp\left(\frac{-\mathrm{j}2\pi ni}{N}\right),\quad n=1,2,...,N. \tag{6}$$

The Fourier descriptor captures only shape boundary information and ignores interior information.

According to the requirement of CAD model recognition, we think that the region information of a 3D model is more important than contour and color information. It is also observed that the Zernike moments descriptor outperforms the Fourier descriptor through experiments on classification tasks. Actually, as shown in Fig. 3, exploiting a hybrid descriptor which integrates the Zernike moments descriptor and the Fourier descriptor does not improve notably the classification accuracy compared with exploiting only the Zernike moments descriptor. Therefore, in this work, the Zernike moments descriptor is chosen to represent the rendered 2D images from 3D models.

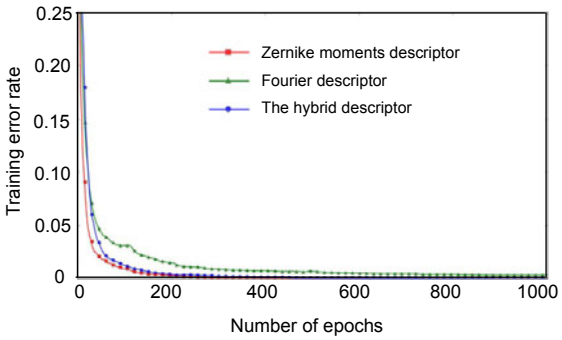

Fig. 3 Comparison of convergence rates among the different descriptors

The steps of extracting features for a 3D CAD model are as follows: (1) translate and scale the 3D model to ensure that it could be entirely contained in rendered images; (2) create 10 light fields for a 3D model; (3) render images from the camera positions of light fields, with 10 images being represented for 20 viewpoints of each light field; (4) extract the Zernike moments descriptor from the rendered im-

ages. These steps are more in accordance with engineers' cognitive habits.

4.3 Input pattern preprocessing

Before providing an input pattern to the deep neural network classifier, the input signal needs to be preprocessed (Bishop, 1995). Considering the trade-off between the precision of shape representation and the computational overhead, only the first 35 Zernike moments are used. The selected LFD is transformed into a vector which has 3500 elements. The input vector can be described as $\{zmd_i\}$, i=1, 2, …, 35, where zmd_i is the ith coefficient of the Zernike moments descriptor. Also, scaling before applying it to deep neural networks is very important. The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because output values usually depend on the activation functions, large attribute values might cause numerical problems. Each attribute of the input vector should be linearly scaled to [−1, 1] or [0, 1]. In this work, the original values of the input pattern are normalized to [0, 1].

5 Construction of the deep neural network classifier

Our deep neural network classifier (Fig. 4) contains five learned layers: an input layer, three hidden layers, and an output layer. The 3D model classification process performed by engineers usually contains three phases: First, they need to distinguish rotational models from non-rotational ones; second, internal and external shape elements are identified separately; third, auxiliary holes and gear teeth are detected. Finally, these key features are synthesized together to form more abstract concepts. Inspired by such a cognitive process which is from shallow to deep, from low-level to high-level, our deep neural network classifier is designed to include three hidden layers.

The constructed deep network classifier is a 3500-28-400-56-28-1 fully connected neural network. It takes in the 3500-dimensional input vectors, and then more abstract features are extracted and synthesized in the hidden layers. If two 3D CAD models belonging to the same class produce feature activation vectors with a larger Euclidean separation at the early layers, we can say that the higher levels of neural networks consider them to be more similar. Finally, the output layer outputs the corresponding model categories.

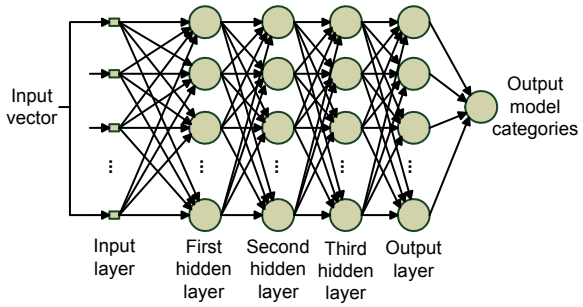

Fig. 4 Architecture of our deep neural network classifier

If the deep neural networks contain a large number of trainable parameters, which have high expression ability, they can be finely tuned to specific training sets. In this situation, however, the testing error rate may be unacceptable, and over-fitting occurs. On the other hand, if the deep neural networks contain a small number of trainable parameters, they will not have enough degrees of freedom to fit the training set well, and error rates on the validation set and test set are still high (Bengio, 2009). Therefore, it is rational to find a trade-off solution. In a maximum likelihood setting, there exists an optimum value of the number of trainable parameters that gives the best generalization performance, corresponding to the optimum balance between under- and over-fitting. According to experience and experiments, we think the optimal ratio between the amount of training data and the number of trainable parameters is about 160:1. Our training set contains 5990 CAD models, and each model is represented by a 3500-dimensional vector, so the amount of training data=3500×5990=20 965 000. The parameter set of the whole deep neural network classifier contains a total of 3500×28+28+28×400+400+400×56+56+56×28+28+28=133 680 trainable parameters, of which 133 168 parameters are the weights, and 512 parameters are the biases. The ratio between the amount of training data and the number of trainable parameters=20 965 000/133 680≈156.8297. This is the main reason why the constructed deep network classifier is a 3500-28-400-56-28-1 fully connected neural network. In our

experiments, we also test the 3500-28-100-28-28-1, 3500-28-300-28-28-1, and 3500-28-300-56-28-1 neural networks. The generalization performances of these neural networks are not as good as that of the 3500-28-400-56-28-1 neural network.

Learning optimal model parameters involves minimizing an error (loss) function. In the case of multi-class classification, it is very common to use the negative log-likelihood as the error function. This is equivalent to maximizing the likelihood of the training set D_{train} under the model parameterized by $\boldsymbol{\theta}$. The likelihood function L is defined as follows:

$$L(\boldsymbol{\theta}=\{\boldsymbol{W},\boldsymbol{b}\},D_{\text{train}})=\sum_{i=1}^{|D_{\text{train}}|}\log(P(Y=c_{\text{model}}^{(i)}\mid \boldsymbol{v}_{\text{model}}^{(i)},\boldsymbol{W},\boldsymbol{b})), \tag{7}$$

where $\boldsymbol{\theta}$ is the set of all trainable parameters for the 3D CAD model classifier, D_{train} denotes the training set, $\boldsymbol{W}$ refers to the weight matrices, $\boldsymbol{b}$ refers to the bias vectors, $c_{\text{model}}^{(i)}$ is the model category of the ith model in the training set, and $\boldsymbol{v}_{\text{model}}^{(i)}$ is the corresponding input vector of the ith 3D model in the training set. The error function E of the deep neural network classifier is defined as

$$E(\boldsymbol{\theta}=\{\boldsymbol{W},\boldsymbol{b}\},D_{\text{train}})=-L(\boldsymbol{\theta}=\{\boldsymbol{W},\boldsymbol{b}\},D_{\text{train}}). \tag{8}$$

This error function is differentiable. The gradient of this function over the training set can be used as a supervised learning signal for deep learning of a 3D model classifier. The topological structure and constituent elements of the deep neural networks are elaborated in detail in the following.

5.1 Input layer

The input layer is decided by the dimensionality of the input vector. To deal with 3D model classification problems, the input vector denoted by $\boldsymbol{v}_{\text{model}}$ is 3500 dimensional after preprocessing. The entire dataset is split into training set D_{train}, validation set D_{valid}, and test set D_{test}. Each data set is an indexed set of pairs $(\boldsymbol{v}_{\text{model}}^{(i)},c_{\text{model}}^{(i)})$, where $\boldsymbol{v}_{\text{model}}^{(i)}$ is the ith training sample in the dataset and $c_{\text{model}}^{(i)}\in\{0,1,...,27\}$ is the category of the ith model $\boldsymbol{v}_{\text{model}}^{(i)}$.

5.2 Hidden layers

Three fully connected neural networks are used as hidden layers in the constructed deep architecture. The layers are referred to as H1, H2, and H3. H1 uses 28 neurons to generate a 28-dimensional feature vector. It has 3500×28+28=98 028 trainable parameters, about 73.33% of the whole network's parameter set.

$$\mathbf{out}_{\text{H1}}=\varphi_{\text{H1}}(\boldsymbol{b}^{(\text{H1})}+\boldsymbol{W}^{(\text{H1})}\boldsymbol{v}_{\text{model}}), \tag{9}$$

where $\mathbf{out}_{\text{H1}}$ is the output vector of the first hidden layer H1, $\boldsymbol{W}^{(\text{H1})}\in\mathbb{R}^{28\times3500}$ is the weight matrix connecting the input vector to H1, $\boldsymbol{b}^{(\text{H1})}\in\mathbb{R}^{28}$ is the bias vector, and $\varphi_{\text{H1}}(\cdot)$ is a nonlinear activation function used in H1.

H2 uses 400 neurons to generate a 400-dimensional feature vector. It has 28×400+400= 11 600 trainable parameters. H2 can be mathematically described as

$$\mathbf{out}_{\text{H2}}=\varphi_{\text{H2}}(\boldsymbol{b}^{(\text{H2})}+\boldsymbol{W}^{(\text{H2})}\mathbf{out}_{\text{H1}}). \tag{10}$$

H3 uses 56 neurons to generate a 56-dimensional feature vector. It has 400×56+56=22 456 trainable parameters. H3 can be mathematically described as

$$\mathbf{out}_{\text{HL}}=\varphi_{\text{H3}}(\boldsymbol{b}^{(\text{H3})}+\boldsymbol{W}^{(\text{H3})}\mathbf{out}_{\text{H2}}), \tag{11}$$

where $\mathbf{out}_{\text{HL}}$ is the output feature vector of the entire hidden layers.

5.3 Output layer

Logistic regression is put on top of the hidden layers as the output layer of the deep network classifier. A single logistic regression layer without combining multiple neural networks itself is a probabilistic, linear classifier (Dreiseitl and Ohno-Machado, 2002). It is parameterized by a weight matrix $\boldsymbol{W}_{\text{LR}}$ and a bias vector $\boldsymbol{b}_{\text{LR}}$. Classification is done by projecting data points onto a set of hyper-planes, the distance to which reflects a class membership probability. The logistic regression layer can be written as

$$\begin{aligned}P(Y_{\text{pred}}&=i\mid\mathbf{out}_{\text{HL}},\boldsymbol{W}_{\text{LR}},\boldsymbol{b}_{\text{LR}})\\&=\text{soft}\max_i(\boldsymbol{W}_{\text{LR}}\mathbf{out}_{\text{HL}}+\boldsymbol{b}_{\text{LR}})\\&=\frac{\exp(\boldsymbol{W}_{\text{LR}i}\mathbf{out}_{\text{HL}}+b_{\text{LR}i})}{\sum_j\exp(\boldsymbol{W}_{\text{LR}j}\mathbf{out}_{\text{HL}}+b_{\text{LR}j})}.\end{aligned} \tag{12}$$

The output of the classifier is then generated by taking the argmax of the vector whose ith element is $P(Y_{\text{pred}}=i|\mathbf{out}_{\text{HL}}, \boldsymbol{W}_{\text{LR}}, \boldsymbol{b}_{\text{LR}})$. It can be computed using Eq. (13), where the output result is denoted by $Y\in\{0, 1, \ldots, 27\}$:

$$Y=\arg\max_i P(Y_{\text{pred}}=i\,|\,\mathbf{out}_{\text{HL}}, \boldsymbol{W}_{\text{LR}}, \boldsymbol{b}_{\text{LR}}). \quad (13)$$

5.4 Nonlinearity and linearity

Each neuron in the deep neural networks has nonlinearity (activation function) and linearity (affine transformation unit). The activation functions selection according to domain knowledge and the weights and biases initialization in linearity are very important to improve the generalization performance of the networks.

The applicable activation function sets should satisfy requirements in terms of nonlinearity, saturability, continuity, smoothness, and monotonicity. The nonlinear activation functions $\varphi(\cdot)$ are generally chosen to be sigmoidal functions such as the logistic sigmoid function and the tanh function. The logistic sigmoid function has the form

$$\sigma(a)=\frac{1}{1+\mathrm{e}^{-a}}, \quad (14)$$

and the tanh function has the form

$$\tanh a=\frac{\mathrm{e}^{a}-\mathrm{e}^{-a}}{\mathrm{e}^{a}+\mathrm{e}^{-a}}. \quad (15)$$

To thoroughly understand the influence of various kinds of activation functions on the deep neural networks, five different schemes are tried in this work (Fig. 5): (1) Use the logistic sigmoid function in all the three hidden layers; (2) Use the logistic sigmoid function in the first two hidden layers and the tanh function in the third hidden layer; (3) Use the logistic sigmoid function in the first hidden layer and the tanh function in the second and third hidden layers; (4) Use the tanh function in the first hidden layer and the logistic sigmoid function in the second and third hidden layers; (5) Use the tanh function in all the three hidden layers.

In terms of training with gradient descent, the logistic sigmoid function is much slower than the tanh function. Therefore, the fifth scheme is selected in our approach.

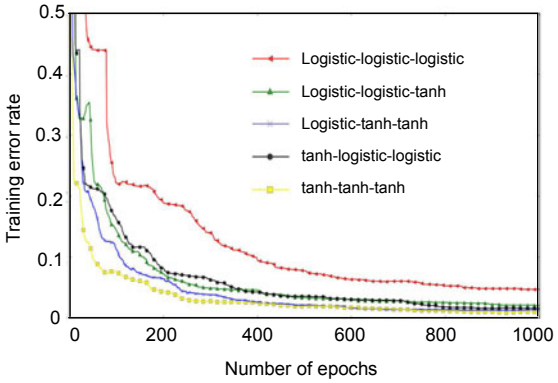

Fig. 5 Comparison of convergence rates among the different nonlinear functions

With respect to linearity of neurons in the hidden layers, one key problem that needs to be considered is weights initialization. With large initial weights, deep networks typically find poor local minima; with small initial weights, the gradients in the early layers are tiny, making it infeasible to train deep networks with many hidden layers. In this study, the initial values for the weights of a hidden layer H_i are uniformly sampled from a symmetric interval that depends on the activation functions. For the logistic sigmoid function, the interval is $\left[-4\sqrt{6/(|H_{i-1}|+|H_i|)}, 4\sqrt{6/(|H_{i-1}|+|H_i|)}\right]$, where $|H_i|$ is the number of computational units in the ith layer. For the tanh function, the interval is $\left[-\sqrt{6/(|H_{i-1}|+|H_i|)}, \sqrt{6/(|H_{i-1}|+|H_i|)}\right]$. This initialization ensures that, early in the training, each neuron operates in a regime of its activation function where information (function signals and error signals) can be easily propagated both forward (activations flowing from inputs to outputs) and backward (gradients flowing from outputs to inputs) (Glorot and Bengio, 2010).

6 Training of the classifier

In essence, training a deep network classifier can be regarded as solving a non-convex optimization problem, because the error function is no longer a

convex function of the model parameters. Hence, applying multiple practical strategies to the training process is needed (Larochelle *et al.*, 2009). We describe some effective strategies adopted in the training phase below. Sections 6.1–6.4 are arranged according to our estimation of their importance, with the most important first. Choosing proper learning rates and training protocols makes learned hyper parameters approximate the optimal solution as far as possible; the early stopping method and 'weight decay' are effective ways to prevent over-fitting (Haykin, 2008).

6.1 Learning rate

A small learning rate is chosen to avoid oscillations, and we use an equal learning rate for all layers. The learning rate η is initialized to 0.13. The largest number of training epochs is set to 2000. The learning rate is tuned linearly as the training process proceeds, by making $\eta=\eta-0.01$ for every one hundred epochs and ends if $\eta\leq 0.02$ (Fang *et al.*, 2005).

6.2 Training protocol

The two most useful training protocols are batch training (steepest gradient descent) and stochastic gradient descent. The batch training protocol uses the whole data set all at once. At each step the weight matrix is moved toward the direction of the greatest decrease rate of the error function; thus, it is also known as the steepest gradient descent. A stochastic gradient descent makes an update to the weight vector based on one data point at a time. This update is repeated by cycling through the data either in sequence or by selecting data points randomly with replacement. The latter handles redundancy in the data much more efficiently. In a stochastic gradient descent, however, once weights update may decrease errors on one single pattern, with increasing the total errors on the entire training dataset (Duda *et al.*, 2001).

The variant protocol used in this work for deep learning is an intermediate scenario in which the updates are based on mini-batches of data points (Ranzato *et al.*, 2010; Ngiam *et al.*, 2011; Bordes *et al.*, 2014). Mini-batch works identically to stochastic gradient descent, except that more than one training sample is used to make each estimate of the gradient. This trade-off reduces variance in the estimate of the gradient, and often makes better use of the hierarchical memory organization in modern computers.

There are no definite rules to choose the mini-batch size S. An optimal S is deep networks-, datasets-, and hardware-dependent, varying from two to thousands. In this work, the mini-batch size S is set to 10 to train the proposed deep network classifier for 3D CAD models; if S is set to 500, the generalization performance of the trained classifier degrades.

6.3 Early stopping method

Splitting the training samples into the training set used for gradient descent and the validation set could combat over-fitting by using the early stopping method. The early stopping method is applied by monitoring the model's performance on the validation set. The training is stopped periodically, and the deep neural networks are tested on the validation set after each training cycle (Prechelt, 1998; Yao *et al.*, 2007). In particular, the periodical training-followed-by-validation process adopted in this experiment is as follows:

After one training cycle (a certain number of epochs), the trainable parameters (weights and biases) are fixed. Then the validation error of each mini-batch in the validation set can be computed.

When the validation phase is finished, another training cycle is restarted. This loop is repeated until an optimal solution is acquired. Algorithm 1 gives the implementation details of the early stopping method.

6.4 Weight decay

Weight decay is a heuristic rule used to control the complexity of deep neural networks in order to avoid over-fitting (Reed, 1993). Weights of networks could be classified into two groups: (1) weights which have a large influence on the networks' performance; (2) weights which have a small or no influence at all on the networks' performance, i.e., the unnecessary weights. Weight decay penalizes those unnecessary weights by addition of a regularization term to the error function. In this work, the error function in Eq. (8) is redefined as

$$\tilde{E}(\boldsymbol{\theta}=\{\boldsymbol{W},\boldsymbol{b}\},D_{\text{train}})=E(\boldsymbol{\theta}=\{\boldsymbol{W},\boldsymbol{b}\},D_{\text{train}})+\lambda R(\boldsymbol{\theta}), \tag{16}$$

where λ is the regularization coefficient, chosen to be 0.0001 in our experiments. If λ is set to 0.0005, the convergence of the training process becomes slow.

The regularizer $R(\boldsymbol{\theta}) = \|\boldsymbol{\theta}\|_2^2$, and $\|\boldsymbol{\theta}\|_2 = \sqrt{\sum_{j=0}^{|\theta|} |\boldsymbol{\theta}_j|^2}$ is the L_2 norm of θ.

Algorithm 1 Early stopping algorithm

Input: vectors and categories of 3D CAD models in the training/validation/test sets, max_epoch, n_train_batches, params (***W***'s and ***b***'s) in networks, patience, improvement_threshold, validation_frequency.

Output: best_validation_losses, best_iter, best_params.

```
1  for epoch ← 1 to max_epoch do
2    for minibatch_index ← 0 to n_train_batches−1 do
3      Modify W, b, and iter;
4      if (iter+1) % validation_frequency == 0 then
5        Calculate validation_losses;
6        if validation_losses < best_validation_losses then
7          if validation_losses < best_validation_losses*
             improvement_threshold then
8          Update patience;
9          end
10         Update best_validation_losses, best_iter, and
             best_params;
11         Calculate test_losses;
12       end
13     end
14     if patience <= iter then
15       Return;
16     end
17   end
18 end
```

7 Experimental results

The proposed approach has been implemented by Python. The IDE is Eclipse, and the Theano library is used (Bergstra *et al.*, 2010). The experiments here are conducted using a single CPU (Intel Quad at 2.66 GHz) with 4 GB memory. After running for 1609.05 min, with 445 epochs, 266 554 iterations, the optimization completes. The deep network classifier for 3D CAD models achieves a test error rate of 1.36%. Fig. 6 shows the training process.

7.1 Quantitative evaluations

Features are extracted from every 3D model in the database as an input pattern. As described in Section 4.2, LFD is used for feature extraction. The average size, vertex number, polygon number, and time used for feature extraction of each model class are summarized in Table 1.

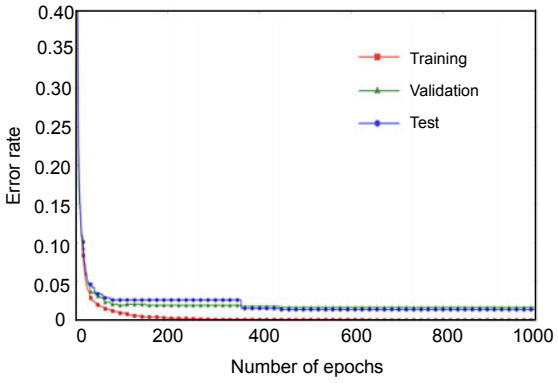

Fig. 6 Curves of the error rates during the training process

Table 1 The average time of extracting features

Index	Category	Avg. vertex number	Avg. polygon number	Avg. time (s)
0	Bearing house	5960.34	11 934.70	2.67
1	Distributor	8353.28	16 719.13	3.05
2	Key	172.04	342.29	0.92
3	Filter	4487.19	8972.97	1.77
4	Sleeve	6794.44	13 571.96	3.71
5	Hook wrench	224.85	445.69	0.48
6	Template	656.72	1329.86	2.87
7	Sealing element	2244.88	4376.37	2.92
8	Lifting hook	7474.63	14 947.18	1.93
9	Tensioner	4444.97	8900.17	1.05
10	Shackle	4588.53	9181.06	2.50
11	Two-end wrench	4938.35	9877.03	1.10
12	Boring bar	2218.75	4448.45	0.91
13	Chunk	1225.72	2447.45	1.85
14	Clamping element	577.49	1080.42	1.34
15	Supporter	833.91	1669.39	3.09
16	Hydraulic part	29 875.50	59 229.79	4.16
17	Valve	6721.18	13 458.31	1.87
18	Mold	20 181.82	41 126.64	6.93
19	Gear	3228.32	6241.43	3.04
20	Washer	721.53	1412.04	1.79
21	Ball	5984.04	11 964.26	4.52
22	Nut	1349.02	2660.03	2.66
23	Screw	4550.94	9067.56	2.04
24	Spring	43 147.67	86 287.29	7.18
25	Wheel	11 956.93	23 921.55	4.44
26	Flange	4610.82	9179.13	3.02
27	Retarder	10 203.35	20 387.00	4.00

The error rate and average time for classification of each 3D model category on the test set are summarized in Table 2. Totally 10 models are wrongly

recognized in the test set. Twenty-three values of error rates are zero. This indicates a good recognition performance for these classes. Some classes like gears have not performed so well. The reason could be that the appearances of these classes vary significantly. The number of training samples being not large enough may be another reason. Compared with traditional classifiers such as SVM, a prominent advantage of the deep network classifier is its high speed of recognition. The average time spent on recognition of all the categories is only about 8.1 ms. The maximum value is 8.7 ms for recognizing sealing elements, and the minimum value is 7.9 ms for recognizing filters, supporters, molds, and wheels.

Table 2 The error rate and average time for classification using the Zernike moments descriptor

Index	Category	Model number	Error number	Error rate	Avg. time (ms)
0	Bearing house	33	0	0	8.1
1	Distributor	22	0	0	8.1
2	Key	192	0	0	8.0
3	Filter	5	0	0	7.9
4	Sleeve	83	0	0	8.2
5	Hook wrench	4	0	0	8.3
6	Template	9	0	0	8.0
7	Sealing element	7	0	0	8.7
8	Lifting hook	14	0	0	8.1
9	Tensioner	19	0	0	8.1
10	Shackle	8	0	0	8.0
11	Two-end wrench	18	0	0	8.0
12	Boring bar	51	0	0	8.0
13	Chunk	7	0	0	8.0
14	Clamping element	9	0	0	8.4
15	Supporter	3	0	0	7.9
16	Hydraulic part	31	0	0	8.3
17	Valve	11	0	0	8.1
18	Mold	7	0	0	7.9
19	Gear	7	2	28.57%	8.0
20	Washer	22	0	0	8.0
21	Ball	23	0	0	8.1
22	Nut	30	0	0	8.1
23	Screw	80	2	2.50%	8.1
24	Spring	16	2	12.50%	8.3
25	Wheel	13	2	15.38%	7.9
26	Flange	8	2	25%	8.1
27	Retarder	5	0	0	8.2
Total		737	10	1.36%	8.1

Some of the visual classification results of the test 3D CAD models recognized by the deep network classifier are shown in Fig. 7. The four columns in the middle show some typical models that are correctly identified by the classifier of the five categories. Fig. 8 shows the other 23 categories that can be recognized a hundred percent.

Fig. 7 The five CAD model categories

The 10 models wrongly recognized by the classifier are listed in the rightmost two columns. The first column gives the five categories that cannot be recognized a hundred percent. The four columns in the middle show some typical models that are correctly identified by the classifier of the five categories

Ten models are not correctly recognized by our classifier. A possible explanation is that these models vary significantly in geometry with respect to their typical models. For example, gear_1 is wrongly recognized as a washer. Tiny thickness and large radius of the inner hole make gear_1 very close to a washer in terms of appearance. gear_2 is wrongly recognized as a flange. Because gear_2 has a hollow cylinder in its axle, it looks like a flange. screw_1 and screw_2 are wrongly recognized as sleeves. flange_1 and flange_2 are wrongly recognized as a washer and a gear, respectively. Another reason for the classification error may be that the input patterns are handcraft features extracted from samples depending on human experience, and some information contained in raw 3D models cannot be well represented by these handcraft features. For example, wheel_1 and wheel_2 are wrongly recognized as sleeves. Because humans recognize the two tires according to their surface texture, the handcraft features adopted in this

work fail to capture such special information. spring_1 and spring_2 are wrongly recognized as a sleeve and a screw, respectively. Through constructing more complex deep learning models which have more hidden layers and providing massive training sample data to these models, the classification accuracy could be further improved by learning features directly from raw 3D data.

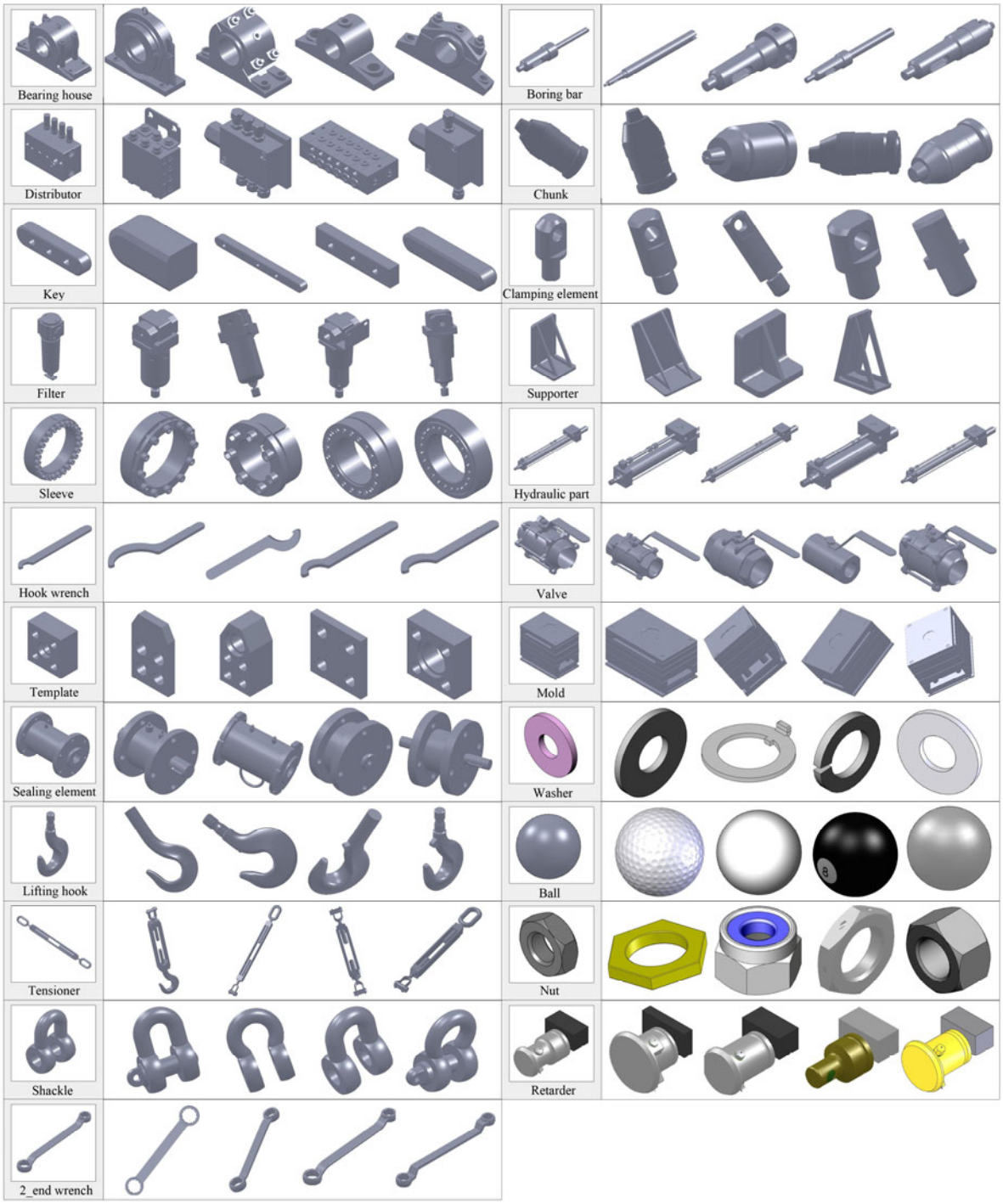

Fig. 8 The 23 CAD model categories recognized with zero error rate
Limited by space, only four models are listed for each category (three models for the supporter)

7.2 Comparison among the different input patterns

The input patterns are important for classification. Besides the Zernike moments descriptor, we run experiments with the Fourier descriptor and the hybrid descriptor to measure the influence of the different input patterns. From the perspective of classification accuracy, the Zernike moments descriptor outperforms the Fourier descriptor. Even if the hybrid descriptor is used which integrates the Zernike moments descriptor and the Fourier descriptor, the classification accuracy cannot be efficiently improved. The dimensionality of the hybrid descriptor, however, greatly increases compared with those of the other two descriptors. The time cost is the highest for recognizing the hybrid descriptor and the lowest for recognizing the Fourier descriptor.

The input pattern of exploiting the Fourier descriptor is a 1000-dimensional vector, which can be described as $\{fd_j\}$, j=1, 2, …, 10, where fd_j is the jth coefficient. The classification results on the test set are summarized in Table 3. Totally 20 models are wrongly recognized in the test set. The error rate on the entire test set is 2.71%. The average time spent on recognition of all the categories is about 4.0 ms.

The input pattern of exploiting the hybrid descriptor is a 4500-dimensional vector, which can be described as $\{zmd_i, fd_j\}$, i=1, 2, …, 35, j=1, 2, …, 10, where zmd_i is the ith coefficient of the Zernike moments descriptor and fd_j is the jth coefficient of the Fourier descriptor. The classification results on the test set are summarized in Table 4. Totally 11 models are wrongly recognized. The error rate on the entire test set is 1.49%. The average time spent on recognition of all the categories is about 9.5 ms.

7.3 Comparison with SVM

The constructed deep neural network classifier is compared with traditional shallow computational architectures such as SVM. The same training set, test set, and input pattern are used for the SVM-based classifier. Only the format of the input vector is transformed to the form which SVM needs. The format of the input vector provided for SVM is {<label> <index1>:<value1> <index2>:<value2>… <index3500>:<value3500>'\n'}, where <label> is an integer indicating the class label, and the pair <index>:<value> gives a feature (attribute) value with <index> being an integer starting from 1 and <value> a real number. Indices must be in ascending order. SVM is a commonly used shallow architecture whose depth is two. SVM implements the 'one-against-one' approach for multi-class classification. The number of categories is 28 in our dataset; hence, 28(28−1)/2=378 classifiers are constructed and each trains data from two classes. The radial basis function (RBF) is chosen as the kernel function, with the best parameter C=32, and gamma=0.001953125. Fig. 9 shows the results. The classification accuracy on the test set achieved by using SVM is 88.47%. Experiments show that the generalization performance of SVM is poorer than those of the deep neural networks.

Table 3 The error rate and average time for classification using the Fourier descriptor

Index	Category	Model number	Error number	Error rate	Avg. time (ms)
0	Bearing house	33	0	0	3.9
1	Distributor	22	2	9.09%	4.1
2	Key	192	0	0	3.9
3	Filter	5	1	20%	4.5
4	Sleeve	83	0	0	4.0
5	Hook wrench	4	0	0	3.9
6	Template	9	0	0	4.2
7	Sealing element	7	0	0	4.0
8	Lifting hook	14	0	0	3.9
9	Tensioner	19	0	0	3.9
10	Shackle	8	0	0	3.9
11	Two-end wrench	18	0	0	3.9
12	Boring bar	51	1	1.96%	4.1
13	Chunk	7	0	0	4.0
14	Clamping element	9	0	0	4.1
15	Supporter	3	0	0	4.0
16	Hydraulic part	31	0	0	4.0
17	Valve	11	0	0	4.1
18	Mold	7	0	0	4.1
19	Gear	7	6	85.71%	3.8
20	Washer	22	2	9.09%	4.0
21	Ball	23	0	0	3.8
22	Nut	30	0	0	3.8
23	Screw	80	1	1.25%	4.1
24	Spring	16	0	0	3.9
25	Wheel	13	5	38.46%	3.9
26	Flange	8	2	25%	4.0
27	Retarder	5	0	0	4.3
	Total	737	20	2.71%	4.0

Table 4 The error rate and average time for classification using the hybrid descriptor

Index	Category	Model number	Error number	Error rate	Avg. time (ms)
0	Bearing house	33	0	0	9.4
1	Distributor	22	0	0	9.5
2	Key	192	0	0	9.5
3	Filter	5	0	0	9.6
4	Sleeve	83	0	0	9.5
5	Hook wrench	4	0	0	10.5
6	Template	9	0	0	9.5
7	Sealing element	7	0	0	9.6
8	Lifting hook	14	0	0	9.5
9	Tensioner	19	0	0	9.7
10	Shackle	8	0	0	9.6
11	Two-end wrench	18	0	0	10.0
12	Boring bar	51	1	1.96%	9.4
13	Chunk	7	0	0	9.4
14	Clamping element	9	0	0	9.3
15	Supporter	3	0	0	9.8
16	Hydraulic part	31	0	0	9.5
17	Valve	11	0	0	9.8
18	Mold	7	0	0	9.3
19	Gear	7	3	42.86%	9.5
20	Washer	22	1	4.55%	9.5
21	Ball	23	2	8.70%	9.3
22	Nut	30	2	6.67%	9.3
23	Screw	80	0	0	9.5
24	Spring	16	1	6.25%	9.7
25	Wheel	13	1	7.69%	9.8
26	Flange	8	0	0	9.8
27	Retarder	5	0	0	9.5
	Total	737	11	1.49%	9.5

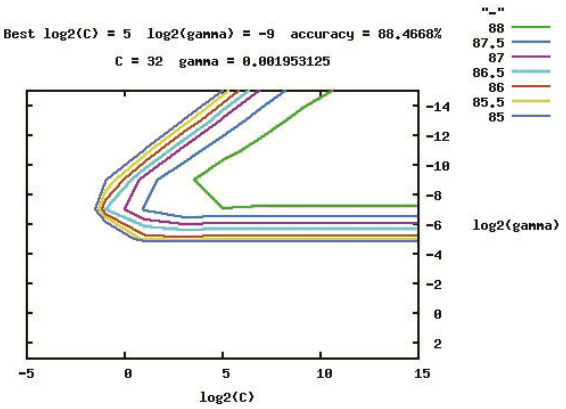

Fig. 9 The classification accuracy on the test set achieved using SVM

References to color refer to the online version of this figure

7.4 Comparison with the existing works

We compare our approach with some previous works (Table 5). The second column shows models contained in the dataset, the third column shows the total number of models in the dataset, and the fourth column shows the total number of categories in the dataset. The fifth column shows the features used to describe the 3D models, the sixth column shows the classifiers, and the average correct rates of each approach are compared in the rightmost column. Experimental results show that our approach outperforms the other approaches.

Table 5 Comparison of our approach and the state-of-the-art approaches

Reference	Model	Total number of models	Total number of categories	Feature	Classifier	Avg. correct rate
This study	3D CAD models	7464	28	Modified LFD	Deep neural network	98.64%
Wu and Jen, 1996	3D prismatic parts	36	–	Simplified skeletons	Back-propagation neural network	–
Ip *et al.*, 2003; Ip and Regli, 2005a	3D CAD models	85	12	Enhanced shape distribution	kNN	72.30%
Ip *et al.*, 2003; Ip and Regli, 2005a	3D CAD models	56	4	Enhanced shape distribution	kNN	66.71%
Hou *et al.*, 2005	3D CAD models	218	6	Moments invariants, geometric ratios, and principal moments	SVM	88.24%
Ip and Regli, 2005b	3D CAD models	100	–	Curvatures	SVM	75.33%
Wei *et al.*, 2008	3D VRML models	30	–	Color	Hopfield neural network	–

8 Conclusions

In this paper, we present an automatic 3D CAD model classification approach based on the deep learning technique. By analogy with the main phases of a manual classification process, a 3D model classifier is constructed based on deep neural networks. Meanwhile, multiple training strategies are properly selected and combined to make the classifier obtain better generalization performance. To the best of our knowledge, we are the first to successfully apply the deep learning technique to 3D CAD model classification. Experimental results are promising. The well trained classifier achieves relatively high classification accuracy on new 3D models. The average time spent on model recognition is sufficiently short.

In the future, we plan to build a larger dataset with much more CAD model categories to further verify the feasibility and effectiveness of the proposed approach and explore its diverse use in real life.

Furthermore, the light field descriptors of 3D models are low level features selected according to prior knowledge of the 3D CAD model classification domain. To completely discard the handcraft features, putting raw 3D model data as input of the deep neural networks is another challenging research direction.

References

Bai, J., Gao, S., Tang, W., *et al*., 2010. Design reuse oriented partial retrieval of CAD models. *Comput.-Aided Des.*, **42**(12):1069-1084. [doi:10.1016/j.cad.2010.07.002]

Barutcuoglu, Z., DeCoro, C., 2006. Hierarchical shape classification using Bayesian aggregation. IEEE Int. Conf. on Shape Modeling and Applications, p.44-48. [doi:10.1109/SMI.2006.15]

Bengio, Y., 2009. Learning deep architectures for AI. *Found. Trends Mach. Learn.*, **2**(1):1-127. [doi:10.1561/2200000006]

Bengio, Y., Courville, A., Vincent, P., 2013. Representation learning: a review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, **35**(8):1798-1828. [doi:10.1109/TPAMI.2013.50]

Bergstra, J., Breuleux, O., Bastien, F., *et al*., 2010. Theano: a CPU and GPU math compiler in Python. Proc. 9th Python in Science Conf., p.1-7.

Bimbo, A.D., Pala, P., 2006. Content-based retrieval of 3D models. *ACM Trans. Multim. Comput. Commun. Appl.*, **2**(1):20-43. [doi:10.1145/1126004.1126006]

Bishop, C.M., 1995. Neural Networks for Pattern Recognition. Oxford University Press, Oxford.

Bordes, A., Glorot, X., Weston, J., *et al*., 2014. A semantic matching energy function for learning with multi-relational data. *Mach. Learn.*, **94**(2):233-259. [doi:10.1007/s10994-013-5363-6]

Chen, D., 2003. Three-Dimensional Model Shape Description and Retrieval Based on Light Field Descriptors. PhD Thesis, National Taiwan University, Taiwan.

Chen, D., Tian, X., Shen, Y., *et al*., 2003. On visual similarity based 3D model retrieval. *Comput. Graph. Forum*, **22**(3):223-232. [doi:10.1111/1467-8659.00669]

Dreiseitl, S., Ohno-Machado, L., 2002. Logistic regression and artificial neural network classification models: a methodology review. *J. Biomed. Inform.*, **35**(5):352-359. [doi:10.1016/S1532-0464(03)00034-0]

Duda, R.O., Hart, P.E., Stork, D.G., 2001. Pattern Classification (2nd Ed.). John Wiley & Sons, New York.

Fang, X., Luo, H., Tang, J., 2005. Structural damage detection using neural network with learning rate improvement. *Comput. & Struct.*, **83**(25-26):2150-2161. [doi:10.1016/j.compstruc.2005.02.029]

Glorot, X., Bengio, Y., 2010. Understanding the difficulty of training deep feedforward neural networks. Proc. Int. Conf. on Artificial Intelligence and Statistics, p.249-256.

Gunn, T.G., 1982. The mechanization of design and manufacturing. *Sci. Am.*, **247**:114-130. [doi:10.1038/scientificamerican0982-114]

Haykin, S., 2008. Neural Networks and Learning Machines (3rd Ed.). Prentice Hall, New York.

Hinton, G.E., Salakhutdinov, R.R., 2006. Reducing the dimensionality of data with neural networks. *Science*, **313**(5786):504-507. [doi:10.1126/science.1127647]

Hou, S., Lou, K., Ramani, K., 2005. SVM-based semantic clustering and retrieval of a 3D model database. *Comput. Aided Des. Appl.*, **2**(1-4):155-164.

Huang, F.J., LeCun, Y., 2006. Large-scale learning with SVM and convolutional nets for generic object categorization. IEEE Computer Society Conf. on Computer Vision and Pattern Recognition, p.284-291. [doi:10.1109/CVPR.2006.164]

Ip, C.Y., Regli, W.C., 2005a. Content-based classification of CAD models with supervised learning. *Comput. Aided Des. Appl.*, **2**(5):609-617.

Ip, C.Y., Regli, W.C., 2005b. Manufacturing classification of CAD models using curvature and SVMs. Int. Conf. on Shape Modeling and Applications, p.361-365. [doi:10.1109/SMI.2005.27]

Ip, C.Y., Regli, W.C., Sieger, L., *et al*., 2003. Automated learning of model classifications. Proc. 8th ACM Symp. on Solid Modeling and Applications, p.322-327. [doi:10.1145/781606.781659]

Iyer, N., Jayanti, S., Lou, K., *et al*., 2005. Three-dimensional shape searching: state-of-the-art review and future trends. *Comput.-Aided Des.*, **37**(5):509-530. [doi:10.1016/j.cad.2004.07.002]

Kavukcuoglu, K., Sermanet, P., Boureau, Y., *et al*., 2010. Learning convolutional feature hierarchies for visual recognition. Proc. 24th Annual Conf. on Neural

Information Processing Systems, p.1090-1098.

Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet classification with deep convolutional neural networks. Proc. 26th Annual Conf. on Neural Information Processing Systems, p.1106-1114.

Larochelle, H., Bengio, Y., Louradour, J., *et al*., 2009. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, **10**(1):1-40.

Ngiam, J., Chen, Z., Koh, P.W., *et al*., 2011. Learning deep energy models. Proc. 28th Int. Conf. on Machine Learning, p.1105-1112.

Prechelt, L., 1998. Automatic early stopping using cross validation: quantifying the criteria. *Neur. Networks*, **11**(4): 761-767. [doi:10.1016/S0893-6080(98)00010-0]

Ranzato, M.A., Mnih, V., Hinton, G.E., 2010. Generating more realistic images using gated MRF's. Proc. 24th Annual Conference on Neural Information Processing Systems, p.2002-2010.

Reed, R., 1993. Pruning algorithms—a survey. *IEEE Trans. Neur. Networks*, **4**(5):740-747. [doi:10.1109/72.248452]

Shilane, P., Min, P., Kazhdan, M., *et al*., 2004. The Princeton Shape Benchmark. Proc. Conf. on Shape Modeling Applications, p.167-178. [doi:10.1109/SMI.2004.1314504]

Wang, W., Liu, X., Liu, L., 2013. Shape matching and retrieval based on multiple feature descriptors. *Comput. Aided Draft. Des. Manuf.*, **23**(1):60-67.

Wei, W., Yang, Y., Lin, J., *et al*., 2008. Color-based 3D model classification using Hopfield neural network. Proc. Int. Conf. on Computer Science and Software Engineering, p.883-886. [doi:10.1109/CSSE.2008.1177]

Wu, M.C., Jen, S.R., 1996. A neural network approach to the classification of 3D prismatic parts. *Int. J. Adv. Manuf. Technol.*, **11**(5):325-335. [doi:10.1007/BF01845691]

Yao, Y., Rosasco, L., Caponnetto, A., 2007. On early stopping in gradient descent learning. *Constr. Approx.*, **26**(2):289-315. [doi:10.1007/s00365-006-0663-2]

Zhang, D., Lu, G., 2002. An integrated approach to shape based image retrieval. Proc. 5th Asian Conf. on Computer Vision, p.652-657.

ESI Journal Ranking: *J ZHEJIANG UNIV-SCI B* Ranks No. 10 in Multidisciplinary

ISI Web of Knowledge℠

Essential Science Indicators℠

WELCOME | HELP | RETURN TO MENU | IN-CITES

JOURNAL RANKINGS IN MULTIDISCIPLINARY

Display items with at least: 0 Citation(s)

Sorted by: Citations SORT AGAIN

1 - 20 (of 25) [1 | 2] Page 1 of 2

	View		Journal	Papers	Citations	Citations Per Paper
1			NAT METHODS	760	59,944	78.87
2			PROC NAT ACAD SCI USA	2,414	43,331	17.95
3			NATURE	571	13,956	24.44
4			SCIENCE	464	12,975	27.96
5			ANN N Y ACAD SCI	1,655	8,027	4.85
6			CURR SCI	1,372	2,861	2.09
7			CHIN SCI BULL	1,275	2,538	1.99
8			J SCI IND RES INDIA	744	2,389	3.21
9			J R SOC INTERFACE	295	2,017	6.84
10★			J ZHEJIANG UNIV-SCI B ★	439	1,693	3.86

Essential Science Indicators was updated on November 1, 2013 to cover a 10-year plus eight-month period, January 1, 2003–August 31, 2013.