Development and Applications of Deep Learning Structures for Point Cloud Data

MAIERDAN MAIMAITIMIN

Department of Intelligent Mechanical Systems
Okayama University

This dissertation is submitted for the degree of
Doctor of Philosophy in Engineering

March 2017

Acknowledgements

This study is done with the help of many great people who contributed in some way to the work described in this thesis.

Firstly, I would like to express my sincere gratitude to my supervisor Prof. Keigo Watanabe for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my whole study. Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Mamoru Minami and Prof. Akio Gofuku, for their insightful comments and encouragement, but also for the hard question which incented me to widen my research from various perspectives. I want to thank Associate Prof. Shoichi Maeyama, Assistance Prof. Isaku Nagai for their supports and suggestions during my Ph.D research.

Every result described in this thesis was accomplished with the help and support of fellow lab-mates. I would like to thank Dr. Tatusya Kato, Dr. Kimiko Motonaka, Ms. Yin Yin Aye, Ms. Yu. Lang, Mr. Shilin Yi and Mr. Mikuria Kota in for the stimulating discussions, research advise, and for all the fun we have had in the last five years.

Last but not the least, I would like to thank my family: my parents Abla Memtimin and Jamal Gulnar, and to my sister Memtimin Dilnar for supporting me spiritually throughout writing this thesis and my life in general, also thanks to my beloved Arken Medine for encouraging and supporting me through my many times of frustration and confusion.

Abstract

This thesis makes three main contributions on a generalized method by using a surface common feature based Deep Learning struture, that amis to acheve object recognition task and emotional facial expression recognition task.

Because the popularity of DL has grown rapidly in the past few years, it leads many research fields to have a qualitative leap, specially in the computer vision field. However, most of the computer vision tasks based on the DL are focused on the 2D image data, and a 2D image is very vulnerable to the changing of environment. On the other hand, the 3D point cloud data(PCD) is the collection of descriptive points of an object in 3D space. It is very activated in distance-based segmentation, object recognition, and also self-driving system. The advantages of PCD are, it able to describe full geometrical information about the objects, and it is not affected by the lighting. But it also has some disadvantaged attributes, such as its high dependency on reference coordinates, thus, its feature appears totally different in the different coordinate system. And, compare to the 2D images, the useful information in 3D PCD is not directly observable. Most of the previous researches on PCD-based methods are only considered the information from the point itself, which is the distance between the point and the observer, or using local feature extraction methods. But eventually some of these research fell into 2D image processing, which causing the information loss, and some other local feature based methods are limited in few objects, which makes these methods are not widely applicable.

As a generalization methods to many problems, DL attempts to model high level abstractions in a complex data source. By changing the structure of network connection, and some techniques such as convolution, pooling, dropout and recurrence, it becomes a very powerful machine learning algorithm. A well-designed DL can avoid the amount of problems, which the deep neural network faced in previously, such as the vanishing of gradient and over-fitting problems. The most important advantage of DL is that the manual design features are not required anymore. So that, DL able to solve a complex problems freely in very high dimension weight space. However, by the same reason, it is very difficult to understand the principles of a DL.

A new DL structure is proposed upon the human touch sensing theory in this thesis, which is able to keep the advantages of PCD and it is based on the generalized features. To be specific, there are two main cells in human touch sensing, one is a texture detection cells named Meissner corpuscles, it responds to the roughness of a surface by the vibration sensing, the other one is Merkel cells, an edge and curvature detection neuron responds to the skin indentation. Under this knowledge and some previous research, a new DL structure as the front end convert network is proposed to simulate these senses. The core approach of this front end DL includes two parts, a stacked memories convolutional autoencoder(sMCAE) is used to simulate the vibration neurons, that obtains the roughness of the surface from a PCD as brain firing map, the other one is used to sense the skin indentations which react to the object edge and surface curvature. To achieve a successful classification, firstly, a supervised methods are used to define some simple classes, such as upward inclined, downward inclined, upward curved, downward curved, edge and flat. In a later progress, an improved skin indentation methods are proposed to produce a surface condition features, which based on the surface normal vectors. In this improvement, the features are no longer designed by manual, a stacked autoencoder is used to extract the features by feeding huge numbers of samples. A properly designed convolutional neural network(CNN) is used as the back end network to merge the roughness information and surface condition features, the it predicts the objects. As a result, the network finally able to achieve 90 percent of accuracy.

In the last part, we trained and applied our final model on two tasks, one is recognition of emotional facial expression, the other one is an object recognition task.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Background

The earlier research of 3D object recognition is mostly based on the model-based algorithm[1–4]. In these research, the point-related feature extraction methods such as Scale-Invariant Feature Transform (SIFT), Histograms of Oriented Gradients (HOG) and Kd-tree, are used to produce the key points from point cloud data (PCD), then these key points are used to match the database, to find the most "likelihood" objects. However, these methods can only be used to recognize the objects whose key features have already been manually included in the model database. Thus, it cannot be applied universally. In order to use the advantage of PCD, whic is able to describe the full geometrical information of the objects. Therefore, many previous researchers have extensivly studied the PCD in neural network (NN) [73, 74], but eventually they reverted back to 2D image processing, which will lose most parts of the information of an object.

On the other hand, in the past few years, the popularity of Deep Learning (DL) has rapidly increased in many fields, leading these fields to qualitative leap, especially in the computer vision. With increasing of computational abilities of computer, and developing of various techniques, the DL becomes more powerful, as found from its ability even beyond the human capacity in some fields, e.g., Google DeepMind (Alpha Go) and Microsoft ImageNet. The DL is a concept, and a class of some modified neural networks. The DL includes two main regions: one is based on the multilayer neural network, and the other is related to restricted Boltzmann machine. DL attempts to model advanced abstractions from complex data. The difference between a deep neural network (NN) and the DL is that the former attempts to convert a complex nonlinear problem into many simple nonlinear problems by only increasing the hidden layers, where as the latter has a deep structure, but with multiple other techniques, i.e., convolution, dropout, pooling and recurrent. By changing the

connection structure, adding convolution layers, processing the pooling operation, dropping out some random nodes and pre-training the parameters, a well-designed DL can avoid various problems which the DNN has conventionally, i.e., gradient vanishing and over-fitting. The other main advantage of DL is that the feature engineering is not required anymore, so that the DL can solve any complex problems in very high dimensional weight space.

1.2 Aims of The Thesis

This research is aimed at combining the advatages of DL and PCD to build a generalized object recognition method. It is based on the common geometrical features of objects, such as the surface curvatures and the surface roughness. In order to achieve this goal, an improved DL structure is proposed by applying a theory of human touch sensing, in which can be transfered the point cloud data in two ways, i.e., the surface roughness and the surface curvatures. Then a general convolutional neural network is used as the final layer to combine these two features. This DL structure is able to achieve a high accuracy in both object recognition and emotional facial expression recognition.

1.3 Contributions

In this thesis, we have three main contributions.

- A Hidden Markov Model based object recognition system by using object's shape information and surface curvature of the object is proposed. Some basic surface curvatures are defined by calculating the vectors between two presented points. An additional Hidden Markov chain is used as the belief network which is based on the surface curvature information to increase the prediction accuracy of the main Hidden Markov chain . This method produces nearly 70 percent of accuracy. The most advantage in this method is that a new feature concept, i.e., a surface curvature, is introduced.

- A DL structure which is able to present the PCD as a brain firing map is proposed. This DL included two main networks as a full front-end network. Corresponding to the two different neuron cells under the human skin, this front-end network includes two NNs: one is a stacked memories convolutional autoencoder (sMCAE), which is designed to simulate the vibration neurons to convert a PCD to the surface roughness by using a new network model, i.e. a short memory network, and the other uses surface normal vectors to convert a PCD to the skin indentation, which reacts to the object

edge and the surface curvature. A semi-supervised method with the definition of some simple classes is used to pre-train the network. The definitions include upward inclined, downward inclined, upward curved, downward curved, edge and flat. As a back-end network, a properly designed convolutional neural network (CNN) is used to merge the roughness information and the surface condition information, and classify some objects. This network is able to achieve 90 percent of accuracy.

- In emotional facial expression recognition task and object recognition task, the surface common features are designed and trained on this surface common feature based network (SCFnet). An unsupervised learning method is applied, in which there are 300 common features on 3000 objects. By using these pre-trained filters, this SCFnet achieves 97 percent of accuracy. A comparison of our model with other related DL models is also held on and object recognition task.

1.4 Thesis Structure

This thesis includes six chapters.

- In chapter 2, we review the reasons for restatement of neural network based models, define the most basic from a perceptron, and explained a well-designed DL can avoid some problems that DNN faced in previously, such as vanishing problems of gradient and over-fitting problems, etc.
- In chapter 3, a vision-based HMM for PCD is proposed to achieve an object recognition task.
- In chapter 4, a neurological theory of human touch sensing is studied and applied. Then, two deep learning structures as the front end convert network are proposed to obtain the vibration features and the surface condition from the PCD.
- In chapter 5, we apply and test our full DL structure in two tasks, i.e., an object recognition and a recognition of emotional facial expressions task.
- In the last chapter, we conclude this full work.

1.5 Publications

This thesis is written upon the following publications.

Journals

1. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "Object Recognition Using a Human-like Vision Analysis System," *International Journal on Smart Material and Mechatronics*, Vol. 3, No. 1, pp. 140–145 (Jan, 2016).

2. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "An Emotional Recognition System for Facial Expression with Surface Common Features," *International Journal on Smart Material and Mechatronics*, vol. 4, No. 1, pp. 192–198 (Dec. 2016).

3. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "Stacked Convolutional Auto-encoders for Surface Recognition based on 3D Point Cloud Data," *International Journal on Artificial Life and Robotics*, (Accepted: 29, Dec. 2016).

Conferences

1. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "Surface-common-feature Descriptor of Point Cloud Data for Deep Learning," in *Proceedings of the 2016 IEEE International Conference on Mechatronics and Automation*, pp. 525–529, Harbin, China (2016).

2. Yu Lang, Maierdan Maimaitimin and Keigo Watanabe, "Design and Implementation of HMM for 3D Emotion Recognition," in *Proceeding of International Conference on Smart Material and Mechatronics,*, pp. 34–37, Makassar, Indonesia (2016).

3. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "Multi-criteria based Facial Expression Recognition by Using Deep Learning," in *Proceedings of the 17th SICE System Integration Division Annual Conference*, pp. 140–144, Hokkaido, Japan (2016).

4. Maierdan Maimaitimin, Yu Lang and Keigo Watanabe, "The Application of Deep Learning on 3D Point Cloud Data," in *Proceedings of Symposium on Fuzzy, Artificial Intelligence, Neural Networks and Computational Intelligence*, Osaka, Japan (2016).

5. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "An Application of ANNs to Human Action Recognition Based on Limbs' Data," in *Proceedings of the Twentieth International Symposium on Artificial Life and Robotics*, pp. 543–546, Beppu, Japan (2015).

6. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "Human Action Recognition Based on the Angle Data of Limbs," in *Proceeding of Joint 7th International Conference on Soft Computing and Intelligent Systems and 15th International Symposium on Advanced Intelligent Systems (SCIS-ISIS)*, pp. 140–144, Kitakyushu, Japan (2014).

7. Maierdan Maimaitimin, Keigo Watanabe and Shoichi Maeyama, "Estimation of Human Behaviors Based on Human Actions Using an ANN," in *Proceedings of 14th International Conference on Control, Automation and System*, pp. 94–98 , Korea (2014).

Chapter 2

Neural Network and Deep Learning

In this chapter, reasons for the restatement of neural network based models are reviewed, i.e., it starts from the definition of the most basic neuron to the explanations of how deep learning methods perform better than ordinary neural network. Simply, By going through the single perceptron, multilayer neural network, convolution neural network and autoencoder, the DL is discussed. This chapter is ended with a short comparison of recognition results in handwriting digits, and the Fig. 2.1 shows the sample of Mixed National Institute of Standards and Technology (MNIST) dataset. However, before we discuss the concept of neural network,

Fig. 2.1 Few samples from the MNIST database of handwritten digits; it has a training set of 60,000 examples, and a test set of 10,000 examples, and the digits have been size-normalized and centered in a fixed-size image

as well, a brief introduction of neurons takes place at the beginning. Secondly, after the simple discussions of perceptions, multilayer Neural Network (NN) is described. It is a most typical supervised learning algorithm and widely used until a decade ago. Depending on the type of the output unit, this model also can be used for the purpose of classification or regression. At the third section, the improved multilayer neural network, i.e., a Convolutional Neural Network (CNN) is described. CNN is specialized in pattern recognition tasks. It is well-known for robustness to small inputs variations, minimal pre–processing and does not

require any specific feature extractor choice. By discussing CNN structure, a clean view on the concept of deep learning is shown. An unsupervised learning algorithm, i.e., autoencoder, also is discussed. In the finally section, a conclusion about the neural network and deep learning is discussed in an example of MNIST dataset.

2.1 Neural Network

To know deep learning, understanding a neural network is the best way. Recently, deep learning becomes a whole new subject, but eventually the deep learning is still a type of neural network which is based on a bio-neural network structure.

2.1.1 Neuron

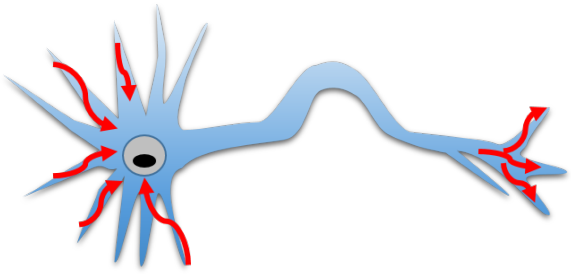

Fig. 2.2 A bio-neuron

In this section, a basic introduction of neural network is described. As the elementary building blocks of the brain and the rest of the nervous system[5, 6], it is very simple. There are several types of neurons in the human brain. A typical neuron is illustrated in Fig. 2.2 with nucleus and all the dendrites. The dendrites are the input to the system while the output can be transferred to the synapses via axon, and synapses are the junctions to other neurons. Roughly speaking there are lots of neurons in human brain, around 100 billion each of which is connected to the average 7000 neurons [7–9]. The received information in a specific neuron (unit), from many different sources (inputs), is integrated into a single real number. This real number is used to show the determination from the input signal to the specialized correspondings. After this evaluation, the neuron outputs something that illustrates the results. This is the famous integrate-and-fire model of neural function [10–13].

The neurons are non-deterministic, and they only emit spikes depending on their activeness, so even with the same input, the output at the other end is random to some extent. The sure thing is that the more there are inputs which excite a neuron, the more spikes are generated per second.

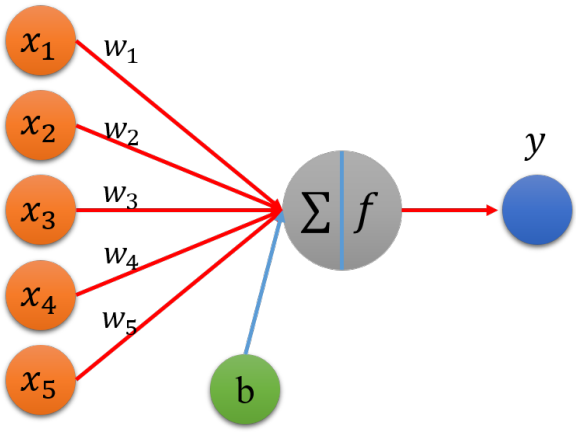

Fig. 2.3 A simple neuron unit with an input-output relation

In NNs [14, 15], the information processing is performed in two stages: one is a linear addition like what mentioned before for natural neurons, and the other is a nonlinear function, which produces the actual output of the unit (neuron), as shown in Fig. 2.3, which is described by

$$y = f(b + \sum_i x_i w_i) \tag{2.1}$$

where b is the bias, x_i is the ith input to the neuron, w_i is the weight of the ith input, $f(\cdot)$ is the nonlinear function, and y is the output.

2.1.2 Activation Function

The activation function is denoted as a determination method for the output of the neuron, where its input is the product of the neuron's input and weight. There are various kinds of activation functions depending on the network structure and problem setting.

Step function

To discuss the improvement of other activate functions, we need to start from the most basic function, the step function. Below, u refers in all cases to the weighted sum of all the inputs to the neuron, i.e. for n inputs,

$$u = \sum_{i=1}^{n} x_i w_i \tag{2.2}$$

The output y of this activation function is binary, depending on whether the input meets a specified threshold, θ. The "signal" is sent, i.e. the output is set to one, if the activation meets the threshold, as described by

$$y = \begin{cases} 1, & \text{if} \quad u >= \theta, \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

Sigmoid function

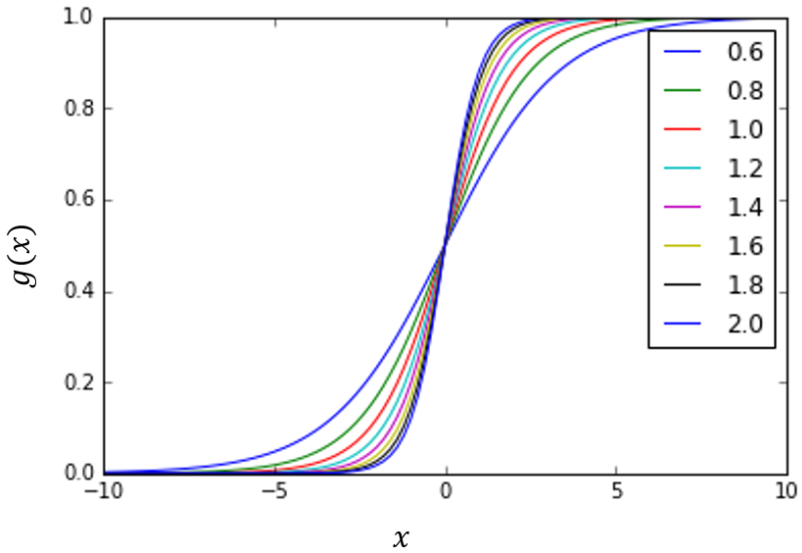

Fig. 2.4 A sigmoid function with different gain

Sigmoid function is a commonly used simple logistic function:

$$g(x) = \frac{1}{1+e^{-kx}} \tag{2.4}$$

where the k is the gain parameter of sigmoid function. The sigmoid function is most attractive as an activation function, because of its monotonicity and simple form, as shown in Fig. 2.4. The activation derivative of sigmoid function with respect to x is obtained by

$$g'(x) \quad = \quad g(x)[1-g(x)] \cdot k \tag{2.5}$$

which means $g(x)$ is increasing monotonously. Also because of its lower derivatives, this activation function is desirable for learning algorithms, such as back-propagation method.

2.2 Multilayer Neural Network and Back-propagation

There are two types of largely classified neural network: a recurrent network and a feed forward network. In this section we focus on the feed forward NN. The feed forward NN has a hierarchical structure consisting of some layers without interconnections between neurons in each layer, and of signals flow from the input-layer to the output-layer in one direction.

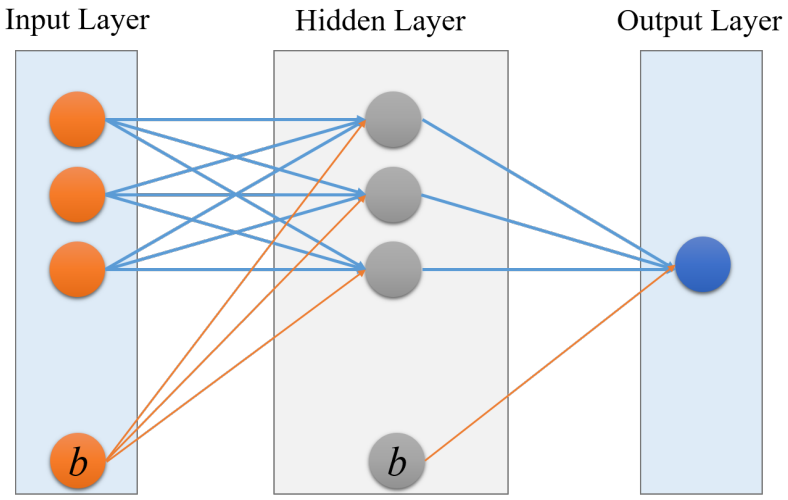

Fig. 2.5 A one output neural network with single hidden layer

Network Formulation

By going through the formulation of a typical MNN, which is shown in Fig. 2.5, it is easy to explain the network content. The circles labeled b are called bias units, and correspond to the intercept term. The leftmost layer of the network is called the input layer, and the rightmost layer the output layer (which, in this example, has only three nodes). The middle layer composed of some nodes is called the hidden layer, because their values are not observed in the training set.

As shown in Fig. 2.5, in our neural network example has three input units (not counting the bias unit), three hidden units in only one hidden layer, and one output unit. As a generic definition, n_t is denoted to the number of layers in our network, thus $n_1 = 3$ in this network. The layer i is labeled as L_i, so that the layer L_1 is the input layer, and L_n is our output layer. The parameters $(W,b) = (W^{(1)},b^{(1)},W^{(2)},b^{(2)})$, where $W_{ij}^{(l)}$ are denoted to the parameter or weight, associated with the connection between the unit j in the layer l, and the unit i in the layer $l+1$. Also $b_{(i)}^{(l+1)}$ is the bias associated with the unit i in the layer $l+1$. Thus, in this case, we have $W^{(1)} \in \Re^{3\times 3}$ and $W^{(2)} \in \Re^{3\times 1}$. Note that the bias b units don't have any inputs

or connections going into them, because they always output the value +1. s_l denotes the number of nodes in the layer l. The $a_i^{(l)}$ is signed to denote the activation of unit i in the layer l. For $l = 1$, also $a_i^{(l)} = x_i$ is denoted to the i-th input. Give a fixed setting of parameters W, b, this neural network defines a hypothesis $h_{W,b}(x)$ that outputs a real number. Specifically, the computation related to what this neural network represents is given by:

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(2)}) \tag{2.6}$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(2)}) \tag{2.7}$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(2)}) \tag{2.8}$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(3)}) \tag{2.9}$$

Now, let $z_i^{(l)}$ denote the total weighted sum of inputs to the unit i in the layer l, including the bias term, e.g., $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)}x_j + b_i^{(2)}$, so that $a_i^{(l)} = f(z_i^{(l)})$.

Note that this easily lends itself to a more compact notation. Once the activation function $f()$ is extended to be applied to vectors in an element-wise way, i.e., $f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$, then the computation is replaced as:

$$z^{(2)} = W^{(1)}x + b^{(2)} \tag{2.10}$$

$$a^{(2)} = f(z^{(3)}) \tag{2.11}$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(3)} \tag{2.12}$$

$$h_{W,b}(x) = a^{(3)} = f(z^{(3)}) \tag{2.13}$$

More generally, we use $a^{(1)} = x$ to denote the values from the input layer, then given the layer l's activations $a^{(l)}$, the layer $l+1$'s activations $a^{(l+1)}$ can be computed as:

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l+1)} \tag{2.14}$$

$$a^{(l+1)} = f(z^{(l+1)}) \tag{2.15}$$

By organizing these parameters in matrices and using matrix-vector operations, we can take an advantage of fast linear algebra routines to quickly perform calculations in a network. So far, we focus on the specified neural network structure. In other words, these equations theoretically show us, that a well designed neural network can solve a complex linear regression or a difficult classification task by increasing the number of hidden layers and output units. For example, Fig. 2.6 shows a multilayer NN, which is capable of classifying

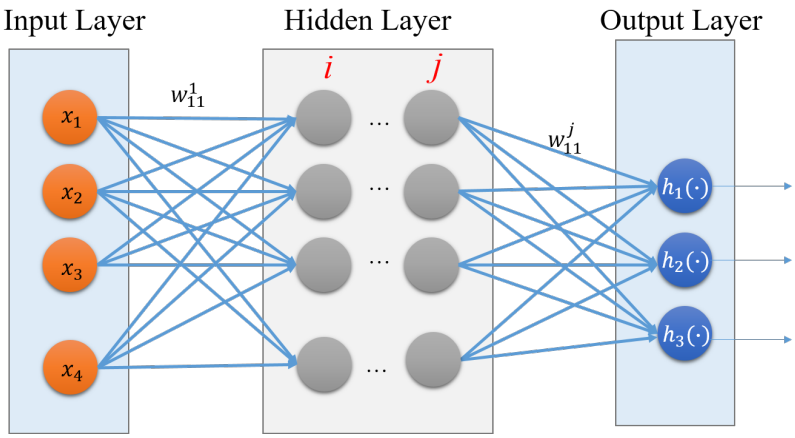

Fig. 2.6 A multilayer neural network

at least three different data (using one-hot encoder), or dealing with a multidimensional regression problem. It works very well at the beginning until the problem becomes more and more complex, but it is discussed later.

Back-propagation Algorithm

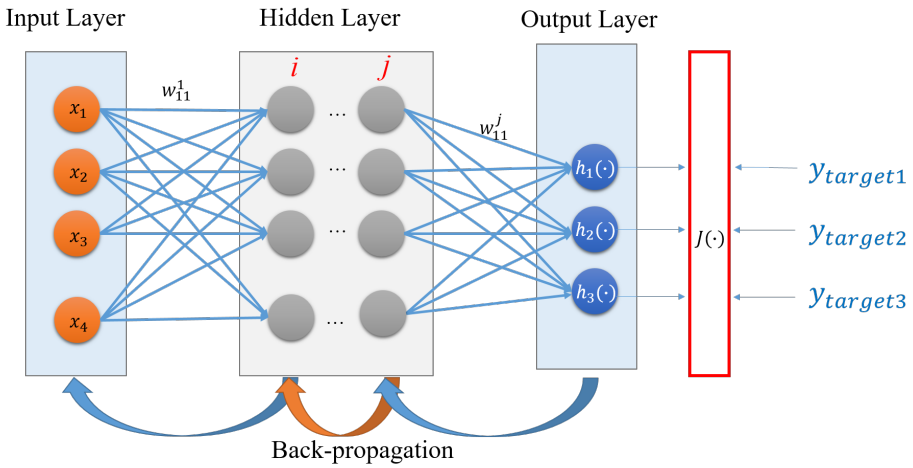

Fig. 2.7 The back-propagation algorithm

The goal of a back-propagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron in the network as a whole. To train a network as in Fig. 2.6, we need to train examples $(x_{(i)}, y_{target} | i \in m)$, where $y_{target} \in \Re^3$ and m is the number of training

examples. For a single training example, i.e., (x, y_{target}), the cost function is defined as:

$$E_{total} = J(W,b;x,y_{target}) = \sum \frac{1}{2} \left\| h_{W,b}(x) - y_{target} \right\|^2 \tag{2.16}$$

The $J(W,b)$ is an average sum of squared error term. To start the back-propagation, we note some more labels on Fig. 2.7. First, let us consider one single weight w'_{ij} which is connected from the first neuron of the last hidden layer to the first neuron on the output layer. The partial derivative of E_{total} with respect to W'_{ij} can be extended by chain rule:

$$\frac{\partial E_{total}}{\partial W'_{ij}} = \frac{\partial E_{total}}{\partial a_j} \times \frac{\partial a_j}{\partial z_j} \times \frac{\partial z_j}{\partial W'_{ij}} \tag{2.17}$$

The derivative of E_{total} with respect to a_j is:

$$\frac{\partial E_{total}}{\partial a_j} = y_j - y_{targer_j} \tag{2.18}$$

and the derivative of the logistic function f with respect to its input u is $f(u)(1-f(u))$, so that,

$$\frac{\partial a_j}{\partial z_j} = a_j(1-a_j) \tag{2.19}$$

where $a_j = f(z_j)$. Next, the derivative of $z^j = \sum_{i=1}^{N} w'_{ij} h_i$ are calculated with respect to a particular w'_{ij} which is simply h_i. Thus, after making the appropriate subsitutions, it becomes:

$$\frac{\partial E}{\partial w'_{ij}} = (y_j - y_{targer_j}) \cdot y_j(1-y_j) \cdot h_i \tag{2.20}$$

With this gradient, the update equation for w'_{ij} can be constructed as:

$$w'_{ij,new} = w'_{ij,old} - \eta \cdot (y_j - y_{targer_j}) \cdot y_j(1-y_j) \cdot h_i \tag{2.21}$$

The gradient of the objective function with respect to the input-to-hidden weights are denoted as w_{ki}. This gradient has already been partially computed when the previous gradient was computed. Therefore, the full gradient is

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j=1}^{M} \left(\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} \cdot \frac{\partial u'_j}{\partial h_i} \right) \cdot \frac{\partial h_i}{\partial u_i} \cdot \frac{\partial u_i}{\partial w_{ki}} \tag{2.22}$$

The sum is due to the fact that the hidden unit to which w_{ki} connects is itself connected to every output unit, thus each of these gradients needs to be taken into account as well. Both

$\frac{\partial E}{\partial y_j}$ and $\frac{\partial y_j}{\partial u'_j}$ have already been computed, so that, it results:

$$\frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u'_j} = (y_j - y_{targer_j}) \cdot y_j(1-y_j) \tag{2.23}$$

Then the remaining derivatives $\frac{\partial u'_j}{\partial h_i}$, $\frac{\partial h_i}{\partial u_i}$, and $\frac{\partial u_i}{\partial w_{ki}}$ are required to be obtained, i.e.,

$$\frac{\partial u'_j}{\partial h_i} = \frac{\partial \sum_{i=1}^{N} w'_{ij} h_i}{\partial h_i} = w'_{ij} \tag{2.24}$$

and again using the derivative of the logistic function

$$\frac{\partial h_i}{\partial u_i} = h_i(1-h_i) \tag{2.25}$$

and finally

$$\frac{\partial u_i}{\partial w_{ki}} = \frac{\partial \sum_{k=1}^{K} w_{ki} x_k}{\partial w_{ki}} = x_k \tag{2.26}$$

After making the appropriate substitutions, we can arrive at the gradient

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j=1}^{M} [(y_j - y_{targer_j}) \cdot y_j(1-y_j) \cdot w'_{ij}] \cdot h_i(1-h_i) \cdot x_k \tag{2.27}$$

Thus, the update equation becomes

$$w_{ki}^{new} = w_{ki}^{old} - \eta \cdot \sum_{j=1}^{M} [(y_j - y_{targer_j}) \cdot y_j(1-y_j) \cdot w'_{ij}] \cdot h_i(1-h_i) \cdot x_k \tag{2.28}$$

2.3 Convolutional Neural Network

In the previous two sections, the basic neural network and the learning algorithm were discussed. These are the most important and kernel objects to any NN structure. With the increase of computational power and the different purposes, the NN architecture has been developed in many different ways, and the Convolutional Neural Network (CNN) [31, 18, 23] is the most famous and effective one.

A CNN is very similar to the ordinary NNs from the previous section: they are made up of neurons that are able to learn the weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a nonlinearity. The whole network still expresses a single differentiable score function, which is from the raw input on one

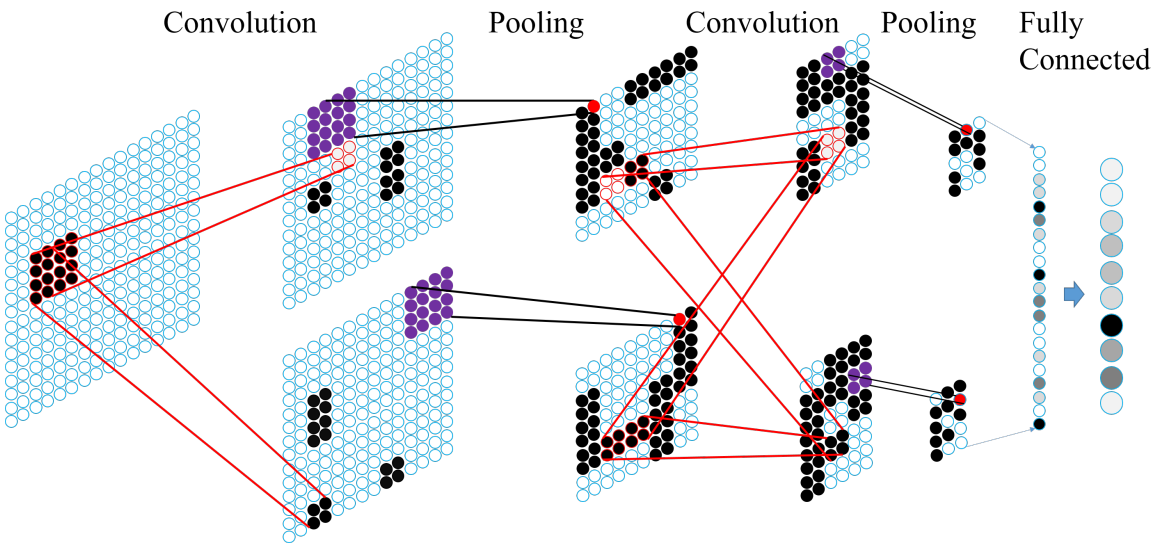

Fig. 2.8 A convolutional neural network with two convolutional layers, two pooling layer, one dense layer, and a softmax estimation layer.

end to class scores at the other. And they still have a loss function (e.g. Softmax) on the last (fully-connected) layer and all the techniques used for learning the regular NN are still applied. To explain the function of CNN, a typical CNN which includes two convolution layers, two pooling layers, and one dense layer is designed as shown in Fig. 2.8, where a black node represents an activated node; a black node with a red shape outline indicates that the current operation contains a valuable node; a red outline node with blank fill represents the deactivated node; and a red node represents the output of pooling operation which is the output of the purple nodes. As this example, in more detail:

- The input layer in the leftmost will hold the raw input values, and the source data are 16×16 matrix in this case.
- The convolution layer will compute the output of neurons that are connected to local regions in the input, each convolution computing is a dot product between their weights and a small region that is connected to in the input volume. This may result in volume such as $[14 \times 14 \times 2]$ if there are two filters.
- The pooling layer will perform a down-sampling operation along the spatial dimensions, which are the width and height, resulting in volume such as $[10 \times 10 \times 2]$.
- The fully-connected layer will compute the class, resulting in volume of size $[1 \times 1 \times 10]$, where each of the 10 numbers correspond to a class.

However, there is a specialized layer, i.e., cross convolution layer, which is shown as the second convolution layer in the figure. In cross convolution, the CNN is able to combine the depth of feature maps. Note that some layers contain parameters, such as the convolution layer and the full-connected layer perform transformations that are a function of not only the activations in the input volume, but also of the parameters. On the other hand, the pooling layers implements a fixed function. The parameters in the convolution layers and the full-connected layers could be trained with gradient descent using or not using pre-training methods. To be more specific, each layer is described in more details.

2.3.1 Convolution Layer

The parameters of convolution layer are related to a set of learn able filters. Every filter is small spatially, along a width and a height, but extends through the full depth of the input volume. For example, as shown in Fig. 2.9, a typical filter on a first layer of a CNN might have the size [3×3×3], where the first and second “3" represents the width and height, and the last one represents the depth of filters (i.e. RGB channel). During the forward pass, each filter is convolved over the input volume and dot productions between the entries of the filters and whole inputs are computed.

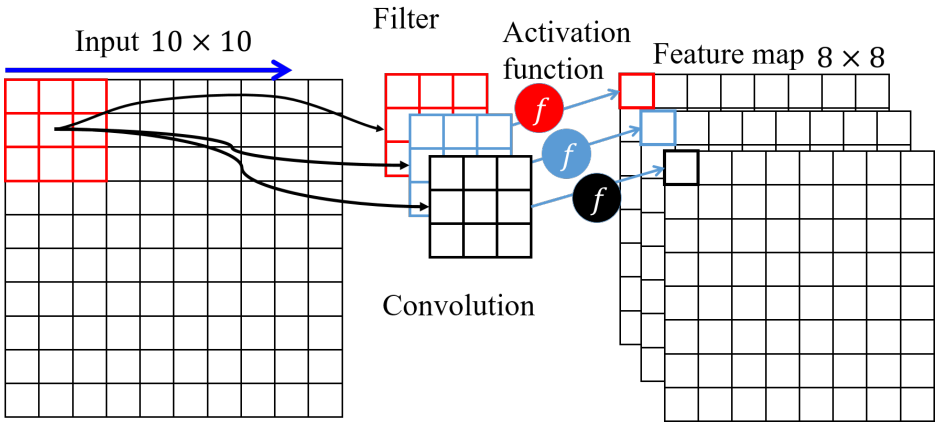

Fig. 2.9 Convolutional operation

At the second convolutional layer, there is a slight difference. In the previous convolutional layer, it produced three different feature maps based on the three different filters. However, these three feature maps also have a neurological connection, we couldn’t just consider them independently. The cross convolutional layer computes dot productions between the entries of the filters and three inputs from the previous feature maps.

With the difference of input, the CNNs can be 2D, 3D or even more dimensional, but what the convolutional layer does is not so much different. The convolutional layer is the

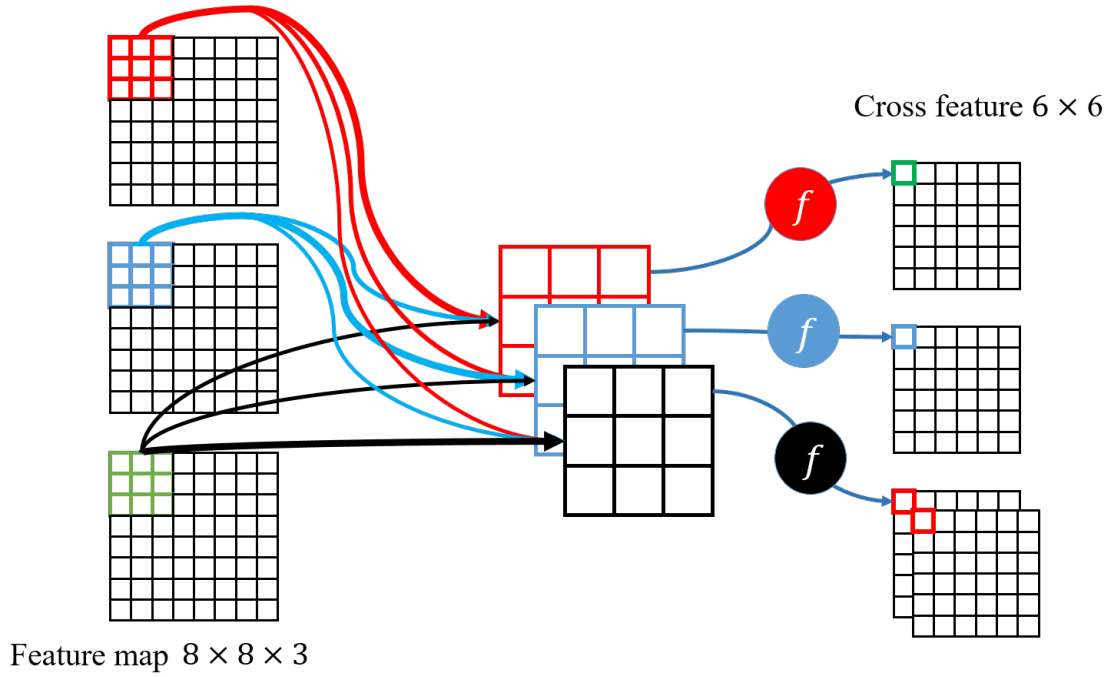

Fig. 2.10 Cross convolutional operation

core building block of a convolutional network that performs most of the computational heavy lifting. In a CNN, a convolution operation is used to extract features from a local neighborhood on the feature maps in the previous layer or inputs. The procedures after the convolutional layer are exactly the same as an ordinary neural network, i.e., the convolutional result is passed with a bias input to the activate function, such as a sigmoid function. Formally, the value of a unit at the position (x,y) in the jth feature map in the ith layer, denoted as z_{ij}^{xy}, is given by

$$z_{ij}^{xy} = Sigm\left(b_{ij} + \sum_{m}\sum_{p=0}^{P_i-1}\sum_{q=0}^{Q_i-1} w_{ijm}^{pq} z_{(i-1)m}^{(x+p)(y+q)}\right) \tag{2.29}$$

where the $Sigm$ is the log-sigmoid function, b_{ij} is the bias for the feature map, m is an index over the set of feature maps in the $(i-1)$th layer connected to the current feature map, z_{ij}^{xy} is the value at the position (p,q) of the kernel connected to the kth feature map, and Q_i and P_i are the height and width of the kernel, respectively.
To summarize a convolutional layer, it follows that

- Accept a volume of size $W_1 \times H_1 \times D_1$
- Require four hyperparameters:

  number of filters K,

their spatial extent F, and

the stride S.

- Produce a volume of size $W_2 \times H_2 \times D_2$, where

  $W_2 = (W_1 - F)/S + 1$,

  $H_2 = (H_1 - F)/S + 1$, and

  $D_2 = K$.

- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.

- In the output volume, the dth depth slice is the result of performing a valid convolution of the dth filter over the input volume with a stride of S, and then offset by dth bias.

2.3.2 Pooling Layer

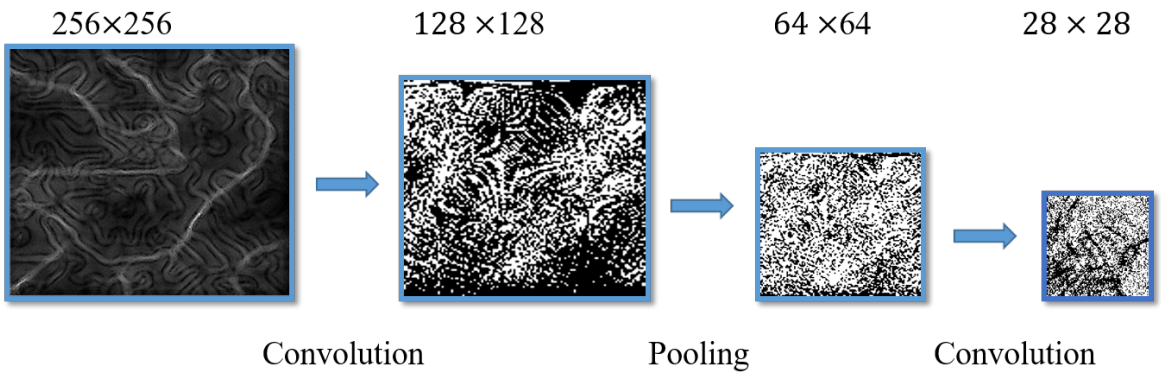

Fig. 2.11 Deep inside of a trained CNN

It is common to periodically insert a pooling layer in-between successive convolutional layers in a convolutional network architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computations in the network, and hence to also control overfitting. There are several types of pooling operations, such as max pooling and average pooling. The max pooling operation is to take a max value over all input values, where as the average pooling is to take the average value over the input values. The depth of the input remains unchanged.
To summarize the pooling layer, it results in

- Accept a volume of size $W_1 \times H_1 \times D_1$

- Require four hyperparameters:

  their spatial extent F and

  the stride S.

- Produce a volume of size $W_2 \times H_2 \times D_2$, where

  $W_2 = (W_1 - F)/S + 1$,

  $H_2 = (H_1 - F)/S + 1$, and

  $D_2 = D_1$.

- Introduce zero parameters because it computes a fixed function of the input.

As shown in Fig. 2.11, after a convolution layer, the input values are turned into a feature map, and the average pooling layer reduces the size of the feature map, and with another convolution layer, the complex inputs are finally turned into a 28×28 feature map, which is a light level feature collection.

2.3.3 Rectified Linear Unit

In the previous section, the log-sigmoid function was mentioned many times, which works very well unless the network structure is getting more and more deep and complex. In a complex NN structure, the gradient descent (GD) does not work as expected, and one of the accepted reason is as follows: once the values reach the log-sigmoid left bottom or right top of the function, the slope is nearly reduced to zero, so that it makes GD not effective anymore. To solve the problem, some researchers proposed a rectified linear unit (ReLu) function [16].

As shown in Fig. 2.12, the ReLU is defined by

$$f(z) = \log(1 + \exp(z)) \cong \max(0, z) \tag{2.30}$$

where

$$z = x_i w_i + b \tag{2.31}$$

There is a very major benefit of ReLU function. It is the reduced likelihood of gradient to vanish when $z > 0$, which means the gradient in this region has a constant value. In contrast, the gradient of sigmoids becomes increasingly small as the absolute value of x increases, which leads to that the gradient of ReLU results in faster learning.

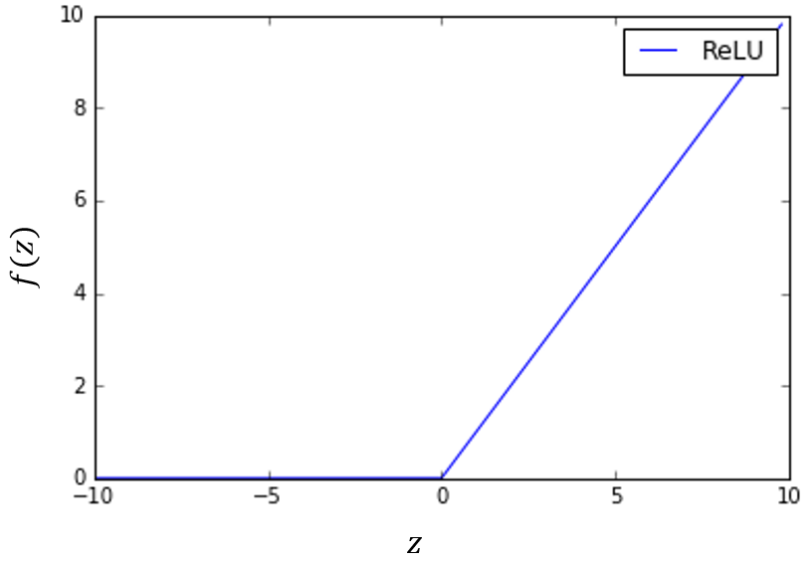

Fig. 2.12 A rectified linear unit

2.4 Autoencoder

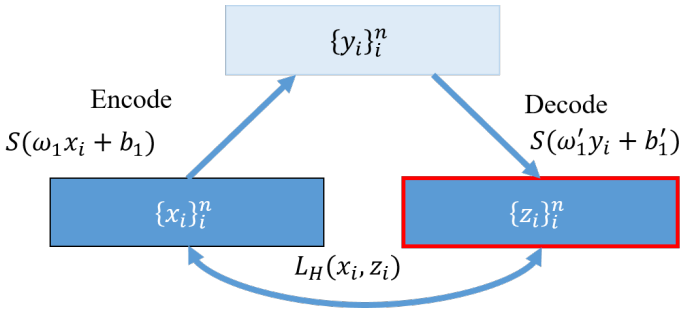

Fig. 2.13 The structure of autoencoder

An autoencoder [32, 19, 33, 22] is an unsupervised learning algorithm, which means it takes an input $\{x_i\}_{i=1}^n$ and first maps it with an encoder to a hidden representation $\{y_i\}_{i=1}^n$ through a deterministic mapping, that is,

$$y = s(\omega x + b) \tag{2.32}$$

where s is a nonlinear activation function such as the sigmoid function. As shown in Fig. 2.13, the latent representation $y_i$is mapped back with a decoder into a reconstruction z of the same shape as x. The mapping happens through a similar transformation, e.g.:

$$z = s(\omega' y + b') \tag{2.33}$$

The weight matrix ω' of the reverse mapping may be constrained to be the transpose of the forward mapping as $\omega^T = \omega'$. This is referred to as tied weights. The parameters ω' and b are optimized such that the average reconstruction error is minimized. And the reconstruction error can be measured in the following function:

$$L_H(x,z) = -\sum_{i=1}^{n} [x_i \log z_i + (1-x_i) \log (1-z_i)] \tag{2.34}$$

where the d denotes the number of inputs.

2.5 An Example

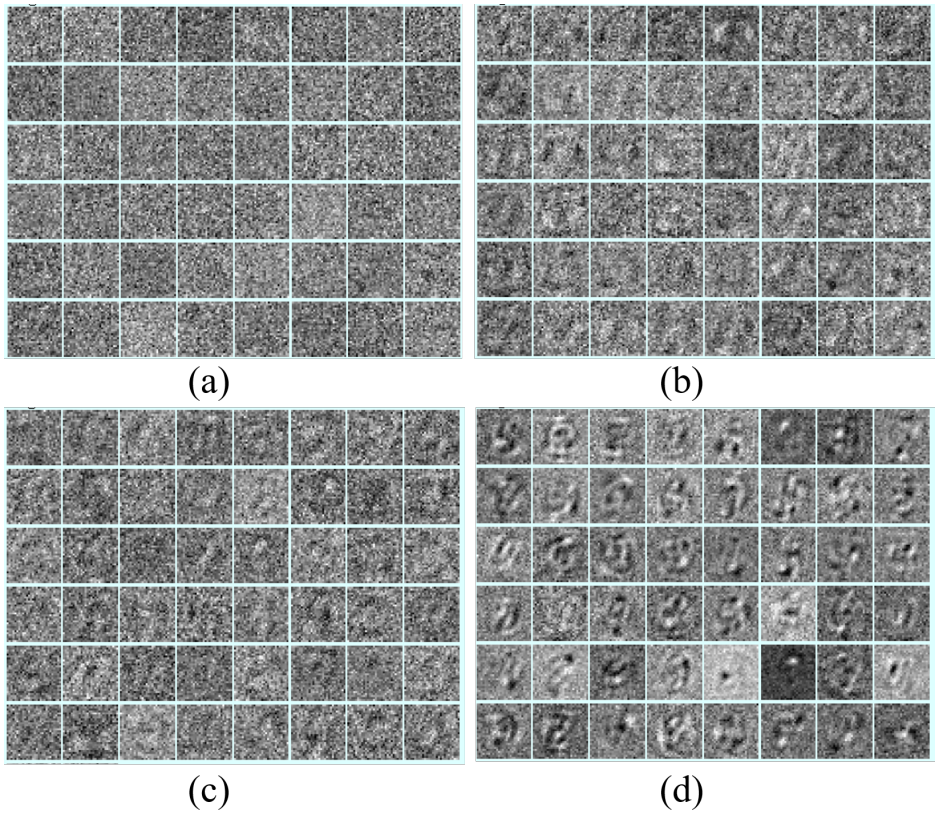

Fig. 2.14 The visualized weights of an autoencoder: (a) after 2000 samples; (b) after 20k samples; (c) after 30k samples; and (d) after 60k samples

The MNIST dataset is applied in a pre-tained Cross-CNN to classify the hand-writing digits. It is a simple and clear classification task. By visualizing the training result and the other parameters, the main adventures of a CNN can be reviewed. Table 2.1 shows the world

accuracy record on the MNIST dataset, and our two simple models, i.e., the Coross-CNN and an autoencoder, are used to compare the results. The autoencoder itself, whose structure is shown in Table 2.2, does not show a good result for this task, but the Cross-CNN model produces an impressive result by using the autoencoder as a pre-training method. We trained a

Table 2.1 The recorded accuracy on the MNIST data and the produced accuracy of our neural network

Name Year	**Error**
DropConect NN 2013	0.0021
McDNN 2015	0.0023
Maxout Network 2013	0.0045
Deep CNN 2015	0.0046
CNN 2003	0.0119
*Cross-CNN	0.0032
*Autoencoder	0.17

*** denotes our model**

simple autoencoder to visulize the weights, which are shown in Fig. 2.14. The reconstruction on the testing data are shown in Fig. 2.15.

Table 2.2 The specified structure of autoencoder

Layer	**Output Shape** Width, Height, Depth	**Parameters**	**Connection**
Input	28, 28, 1	0	
FC 1	1, 1, 50	50×784+50=39250	Input Layer
FC 2	1, 1, 50	50×50+50=2550	FC 1
FC 3	1, 1, 25	25×50+25=1275	FC 2
FC 4	1, 1, 50	25×50+50=1300	FC 3
FC 5	1, 1, 50	50×50+50=2550	FC 4
FC 6	1, 1, 784	784×50+784=39984	FC 5

2.6 Summary

In this chapter, as a prerequisite for understanding the deep learning, the neural networks was discussed. Due to the inclusion of some new algorithms, such as ReLu, convolution and pooling, as well as independent expert filters, that, a deep structured NN, i.e., DL, can learn very complex features from input data. As a comprehension example, we used the MNIST dataset to test two models, i.e., an AE and a Cross-CNN.

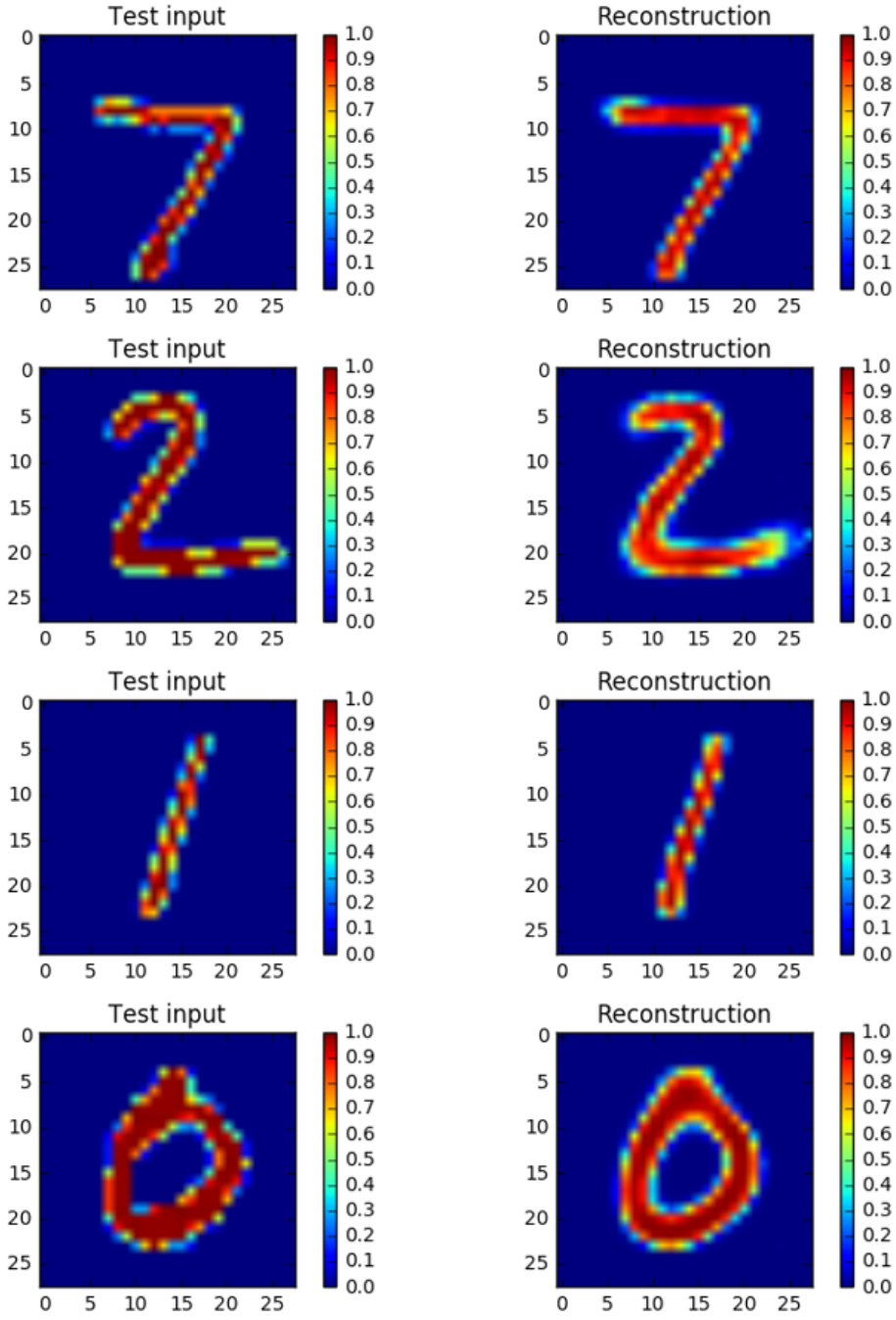

Fig. 2.15 The samples of input and the reconstruction output of the MNIST data.

Chapter 3

3D Point Cloud Data on Deep Learning

In the previous chapter, some models in the frame of both Neural Network (NN) and Deep Learning (DL), which were most related in our research, were discussed. Up to now, there are more than 100 network models, and the number is updated daily. All these networks are developed according to the different data structure. To design a reasonable DL structure with point cloud data (PCD), the understanding of 3D point cloud is very important. This chapter presents the 3D PCD that are going to be studied in vision-based methods.

3.1 3D Sensors

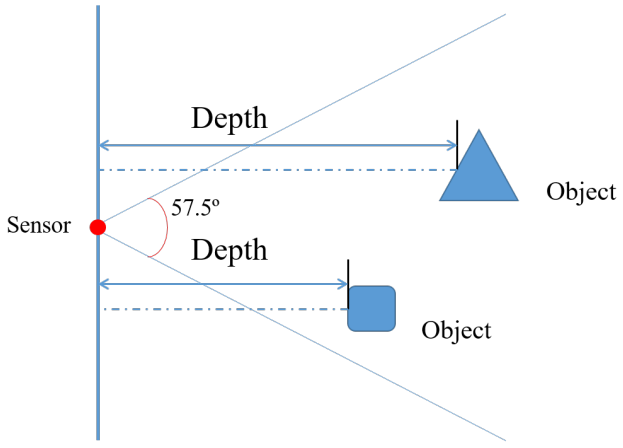

Fig. 3.1 An illustration of the depth stream values.

Understanding the sensor structure and its capability are important, because the data definitions and the devices are directly linked. In this section, the Microsoft Kniect sensor is introduced. The Kinect sensor has been developed and patented under a project *Natal*

since 2006 by Microsoft Company. The Kinect device is primarily based on a depth sensing technology that consists of an Infra–Red (IR) camera and an IR emitter positioned in a certain distance. Although, it includes many other sensors, such as RGB camera and voice sensor, the depth sensor is the main topic in this research.

The depth data from the Kinect camera are provided by the depth stream. The depth data are represented as a frame made up of pixels that contain the distance in millimeters from the camera plane to the nearest object, as illustrated by Fig. 3.1. The depth frame is available in different resolutions. The maximum resolution 640×480 pixel is used in this research. The depth frames are captured in 30 frames per second.

3.2 Point Cloud Data

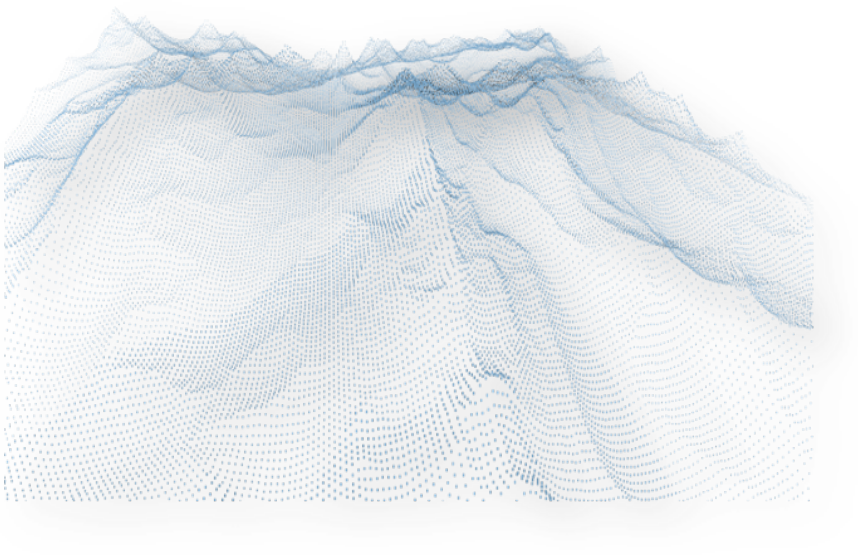

Fig. 3.2 An illustration of the point cloud data.

Since the PCD is not directly captured by Kinect, it needs to use a conversion between the depth and PCD. The depth stream in Kinect is represented by the Z values in uniformed (X,Y) position. In order to obtain a successful conversion, the focal length (FL) and optical center (OC) of Kinect need to be mentioned, where the FL is $(F_x,F_y) = (262.5,262.5)$ and OC is composed of $(C_x,C_y) = (159.75,110.75)$. The following equations represent the conversion

relationship between the depth and PCD,

$$\begin{aligned} x_i &= (u_i - C_x) \times D_i F_x \\ y_i &= (v_i - C_y) \times D_i F_y \\ z_i &= D_i \end{aligned} \tag{3.1}$$

where x_i, y_i and z_i are the x, y and z values of the ith point; u_i is the x pixel position of the ith depth point; v_i is the y pixel position of the ith depth point; and D represents the depth value. By converting the full depth stream, the PCD can be obtained.

A point cloud [36] corresponds to a set of data points in a coordinate system. The PCD in 3D coordinate system can represent a set of points with (x, y, z) values. The PCD includes the geometrical information, which are the shape, surface condition and so on. An information source like PCD, should be very activated in machine learning field, but the truth is opposite. It is very difficult to successfully apply the PCD in machine learning algorithm. One of the reasons for this is that a PCD designates very different features in different reference coordinates. Thus, the features are not stable for a machine learning. On the other hand, the 2D images, they can be represented by a unified measurement method, such as RGB channel, grey level or others. Moreover, each point value in the PCD is very independent, there is no any logical connection between them. Because of these two reasons, when the PCD is directly used in a general NN structure, the network can not converge. To examine this conclusion, a confirmatory experiment is held. Two CNNs with the same architecture are trained, but one of them is trained on 2D images (MNIST), and the other one is trained on PCD. The Fig. 3.3 shows the trained result, in which the CNN trained with PCD does not converge after it reaches 50th epoch.

3.2.1 Vision Based PCD

Intuitively, the PCD is just a set of points in a space, and it is only associated with its reference space. To apply a PCD in any application, the choice of reference is the most important step, before considering it in any machine learning algorithm.

To apply the 3D information, some early researchers proposed some transforming methods, as shown in Fig. 3.4, where the original PCD with three different noise values, the depth map, and the normal-values color map are presented in Fig. 3.4 (a), (b) and (c), respectively. The depth map considers the distance between the points and turns it to the grey level images. The grey images are very good at the distance performance. However, it performs barely active during the changes of small distance. On the other hand, the normal vector colorized

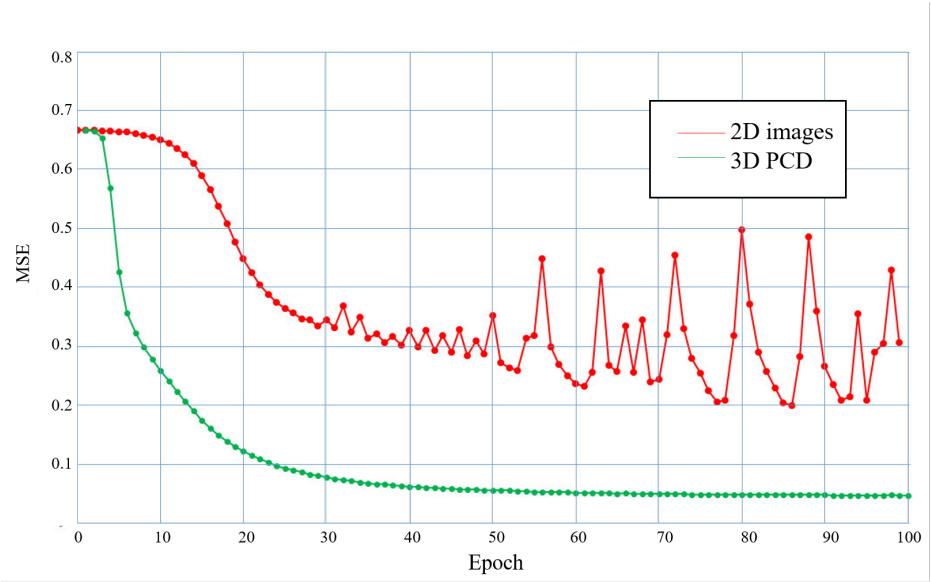

Fig. 3.3 The concept vision of a curvature and a flat plane with edge

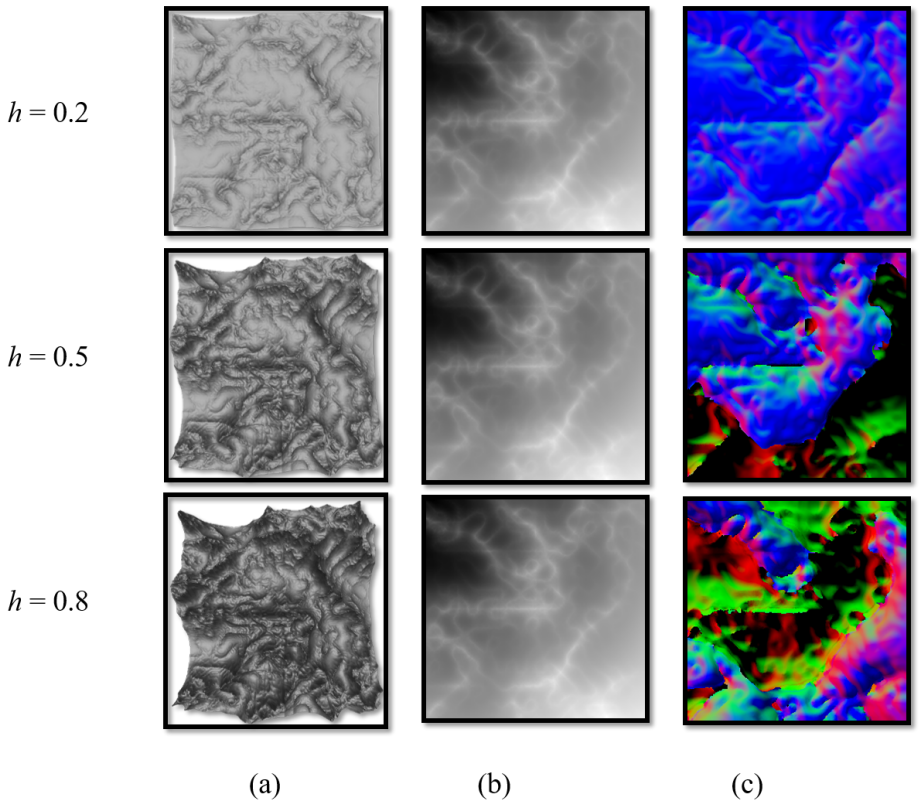

Fig. 3.4 The visual-based transform methods of PCD; (a) is the original point cloud data with three different noise levels; (b) is the depth map transfer method; and (c) is normal vector colorized transform method.

method is very good at sensing the small changes on the surface, but the data dimension is increased to six, which makes it difficult in application.

3.2.2 An Human-like Vision Analysis

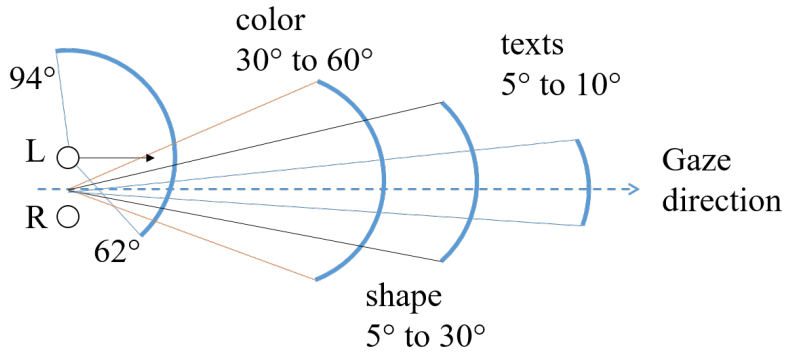

Fig. 3.5 A concept of human vision field

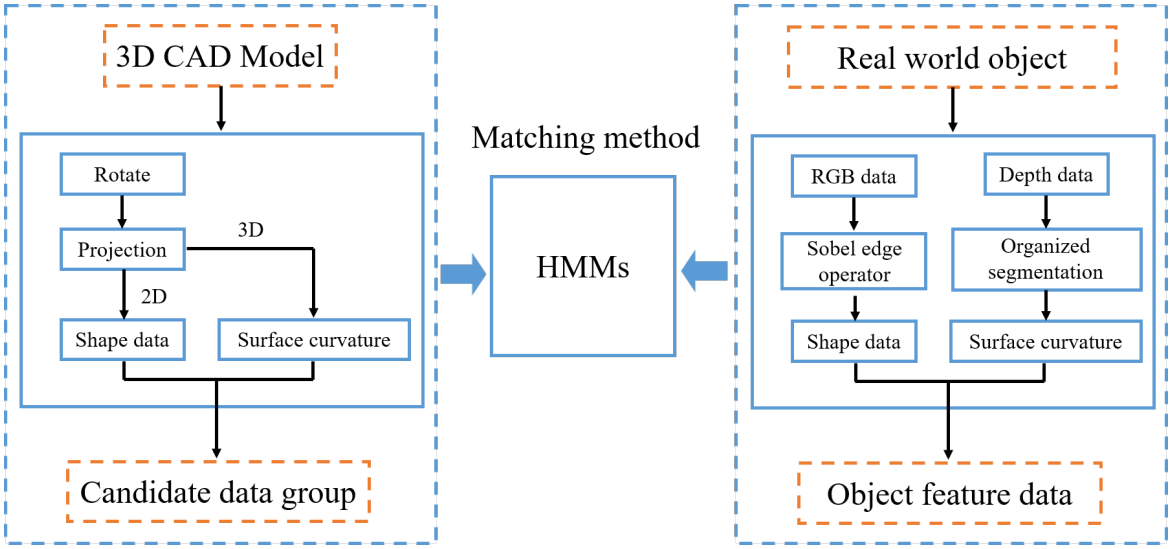

Fig. 3.6 CAD model based recognition system

As shown in Fig. 3.5, an object is visualized by its shape [38, 40], color, and texts for human eyes. Therefore, the 2D projections of 3D PCD are obtained to capture the visualized features. An object recognition system is approached in the study, which is based on the visualized features of the CAD models, and an HMM is applied as the prediction method, where the whole concept is shown as Fig. 3.6.

Fig. 3.7 The object of CAD models

These CAD models are chosen as database because their true objects are easy to be accessed in real world, such as soft drink bottle, tea cup, and tea pot, as shown in Fig. 3.7.

Shape generation

To generate the shape configuration, 3D to 2D projection is indeed[40, 44]. Fig. 3.8 shows the concept of virtual omni-directional projection space. For rotating an object in 3D space, a basic rotation matrix is applied, which is presented in the following equations:

$$R_x(\theta_x) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{bmatrix} \tag{3.2}$$

$$R_y(\theta_y) = \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{3.3}$$

$$R_z(\theta_z) = \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

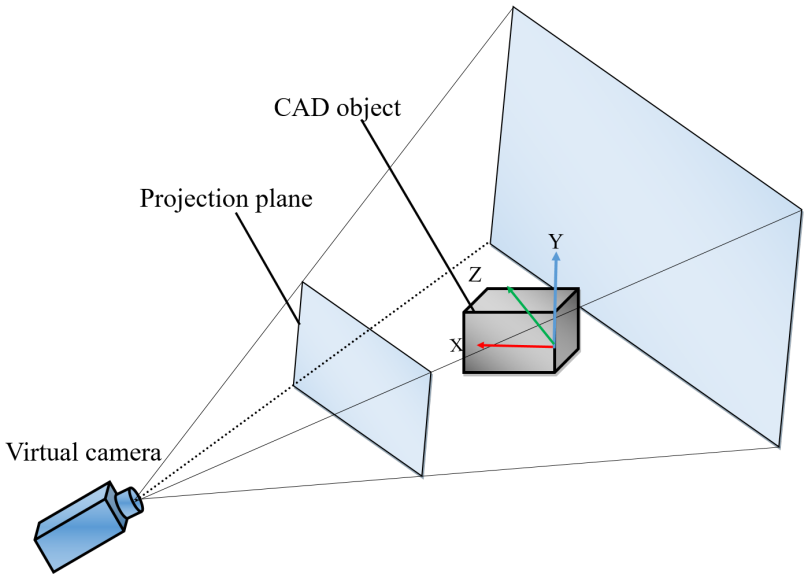

Fig. 3.8 Omni-directional projection

They can be summarized as $R = R_x(\alpha) \cdot R_y(\beta) \cdot R_z(\gamma)$ that depends on the roll angle α, the pitch angle β, and the yaw angle γ. The reason to consider the omni-direction because of the projections of a 3D object in different 2D planes have many different shapes. After rotating the object in each angle, it is projected on a 2D plane to generate the shape. The following equations show a very general way to project the objects onto 2D plane:

$$X' = \{(X - X_c) \times (F/Z)\} + X_c \tag{3.5}$$

$$Y' = \{(Y - Y_c) \times (F/Z)\} + Y_c \tag{3.6}$$

where the virtual camera is located in $(X_c, Y_c, Z_c)$and the 3D point to be projected is denoted as $P = (X, Y, Z)$. The distance from the camera to the 2D plane to be projected is F, so that the equation of the plane is $F = Z - Z_c$. The 2D coordinates of P projected onto the plane are given by (X', Y'). Since the binary shape images of each angle are obtained, as shown in Fig. 3.9, the Sobel operator can be used to generate the edge lines of each projection. Also, the Harris corner detection algorithm is used with a high threshold value to detect a large angle corner. So far, the shape and edge information of a PCD model is obtained, but the texts property is necessary information.

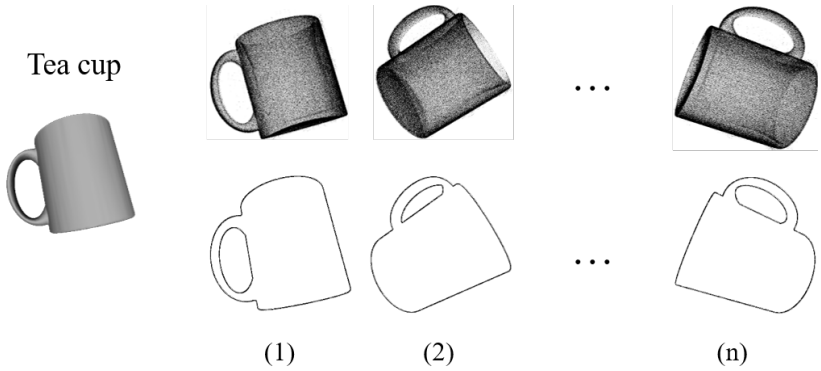

Fig. 3.9 A tea cup shape projection from PCD

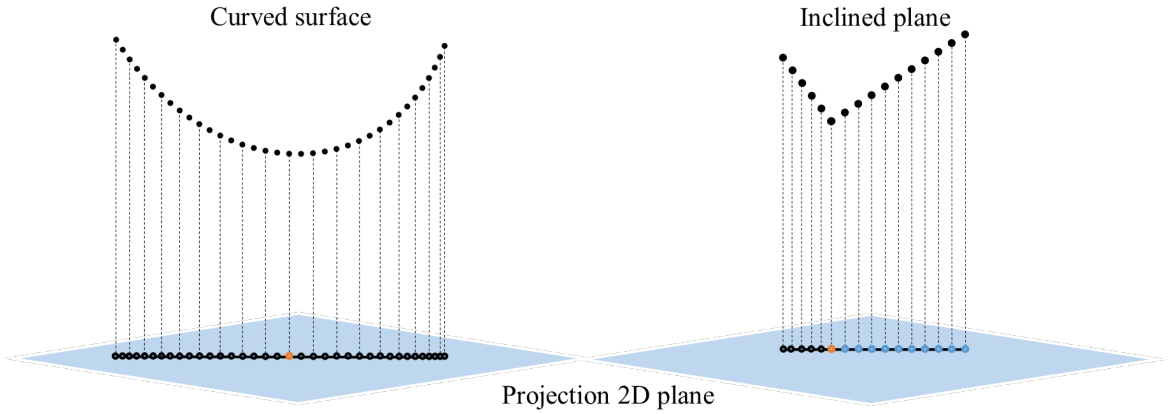

Fig. 3.10 The 2D projection of a curvature and a flat plane with edge

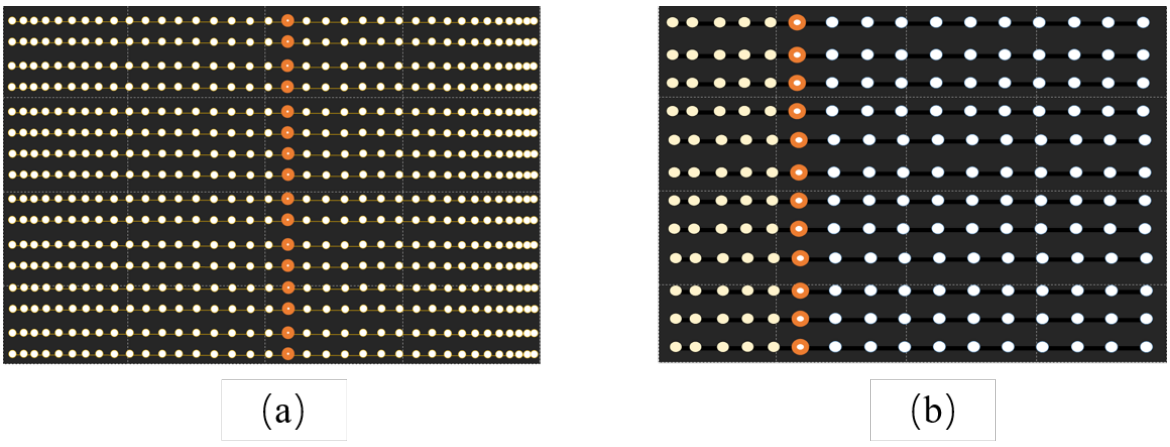

Fig. 3.11 The concept vision of a curvature and a flat plane with edge

Surface analysis

When projecting a point in 3D space onto 2D plane, the curved surface and the inclined plane have different expression in the 2D plane, as shown in Fig. 3.10 and Fig. 3.11 shows the extended visualization. There are plenty of points in each small window, as shown in Fig. 3.11 (a) and (b). The difference between the sizes of two adjacent vectors, is the most important condition to determine the logical value for every plane which is represented by different vectors. The feature vector Δp and $\Delta\theta$ can be defined as:

- If the Δp is changing linearly and $\Delta\theta < 90$, then it is a curved surface without a peak, assigning a value as 1.
- If the Δp is always 0 and $\Delta\theta < 30$, then it is a inclined plane, assigning a value as 2.
- If the Δp is changing and $\Delta\theta < 90$ in different direction, then it is a curved surface with a peak, assigning a value as 3.
- If there isn't an available $\vec{p}$ or the number of $\vec{p}$ is not enough to reflect the changing of Δp, then assigning as 0.

Matching and prediction

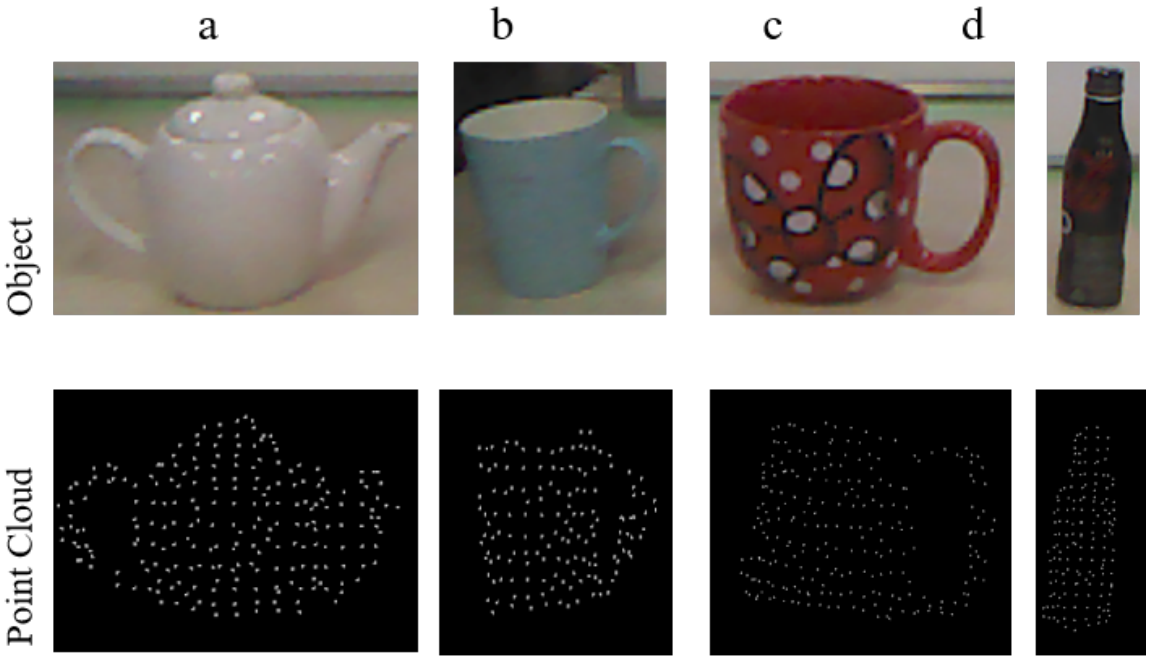

Fig. 3.12 Some samples of selected objects.

So far, the feature data of some objects are obtained from both the 3D CAD models and the real world object, and the real model of objects are shown in Fig. 3.12. For the experiments, a Hidden Markov Model (HMM) based recognition system is approached, and k-nearest neighbor (KNN) method is used to produce the clustering results of the features. The KNN is used to calculate the most nearest matching features in database to the current detected features. The HMM predicts the current detected object by calculating the most likelihood sequence of 16 separated windows' values. In our research, the KNN is not the main topic but the HMM is one of the key relevant research.

The HMM is a well-known probabilistic model of sequence data. In general definition, there are some main parameters:

- a set of N states $\{1,\ldots,N\}$
- an alphabet of M output symbols, $s = \{s_1,\ldots,s_M\}$
- a set of output probabilities, $B = \{b_{ij}|1 < i < N, 1 < j < s_1,\ldots,s_M\}$
- a set of transition probabilities, $A = \{A_{ij}|1 < i < N, 1 < j < s_1,\ldots,s_M\}$
- an initial start probability distribution, $\pi = p_i,\ldots,p_n$

where $N = 3$, i.e., "cup", "teapot", "bottle", B_α is a matrix, which represents the probability of features such as "None", "corner edge", "curved line" and "line". B_β represents the probability of features, i.e., "None", "curvature", "curvature with peek" and "inclined plane". A is 3×3 matrix in our case. To calculate B and A, the dataset includes Cup, Teapot and Bottle, which are generated by using CAD models. The data obtained in three different situations (upright, lateral and inverted) were generated in 36 angles (every 10 degree in Y-Axis). All these data are labeled by their feature vector, which includes 32 elements, 16 digits for shape and surface features. There is one major difference between a general HMM and our specified HMM. In this case, we actually have two feature sets, i.e., the shape feature is captured from RGB and the surface condition is generated from PCD. These two features can not be described in same probabilistic distribution because the shape feature and the surface condition are not relevant at all. So another Markov chain is designed as a belief chain to associate the main Markov chain, which is shown in Fig. 3.13.

For the detail, these two hidden Markov chains have their own tasks, and they separately predict the current piece of feature mostly like object "A", but when the belief chain outputs a prediction as "A" in status T_i, how much trustworthy it is for main chain at T_i moment is denoted as the trust value ω_i to the main chain. In order to calculate the ω_i, we need the sample test data, in which it includes 100 samples in each object. So, the main chain

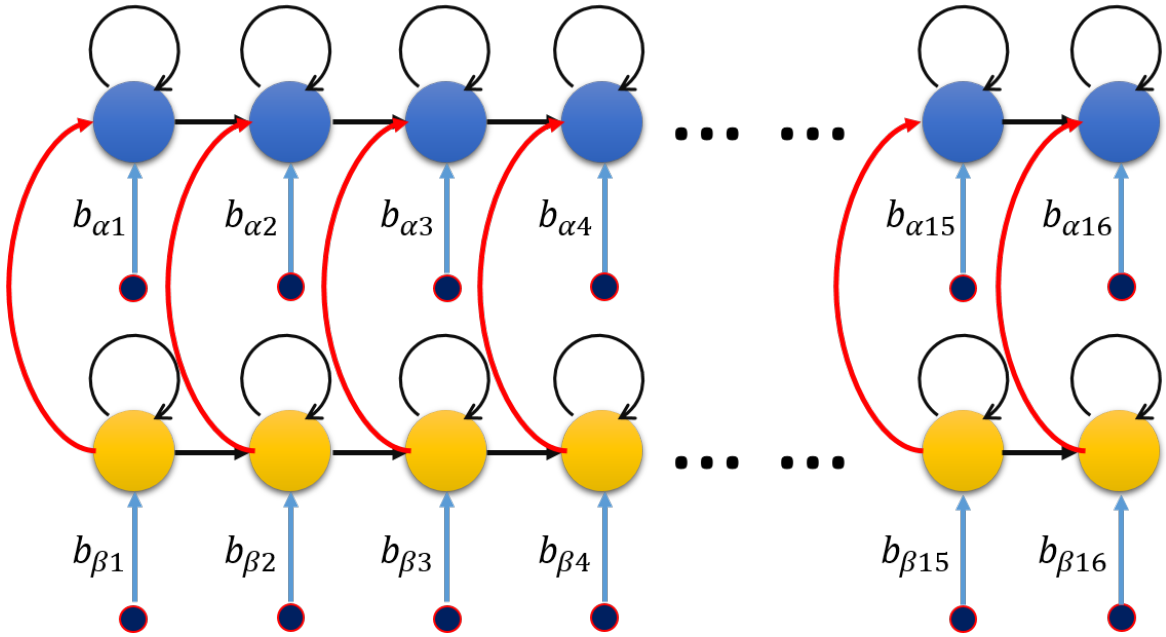

Fig. 3.13 HMM with a belief chain where black line represents the state transition; the blue line represents the output probabilities from observations, which are the small black circle; and the red line is the trust value from the belief chain.

predictions depend on not only the observation, but also the belief chain. where the belief chain depends on the observation O_t and the previous status S_{t-1}.

$$P(\cdot) = P(S_{1:t}, O_{1:t}) = P(S_1)P(O_1|S_1)\prod_{t=2}^{T} P(S_t|S_{t-1})P(O_t|S_t) \tag{3.7}$$

where the $P(\cdot)$ is the prediction probability of current tth status. So that the $P(S_{1:t}, O_{1:t})$ represents the current status probability depends on all the state probabilities up to now. Before calculating the trust value, let the current feature be noted as ω, features x in cup, teapot and bottle be denoted as X, Y, and Z, then the trust value can be factorized as:

$$\begin{aligned} \omega_x &= P(\omega | x \in \{X, \quad Y, \quad Z\}) \\ &= \frac{P(x|\omega)}{P(\omega)} \end{aligned} \tag{3.8}$$

This calculates the real possibility of the belief chain to recognize the right object. However, the main chain not only depends on the observation, O_t and the previous status S_{t-1}, but also depends on the trust values $W(\omega)$ from the current outputs of the belief chain, where the

belief function is designed as:

$$W(\omega_x) = \begin{cases} {\omega_x}^3 & \text{if} \quad 0 < \omega_x < 0.5 \\ 1.2\omega_x - 0.5 & \text{if} \quad 0.5 \leq \omega_x \leq 1 \end{cases} \tag{3.9}$$

Therefore, the final prediction should be:

$$Prediction = \frac{P(S_{1:t}, O_{1:t}) + W(\omega_x)}{P(S_{1:t}, O_{1:t}) + \omega_x} \tag{3.10}$$

Result

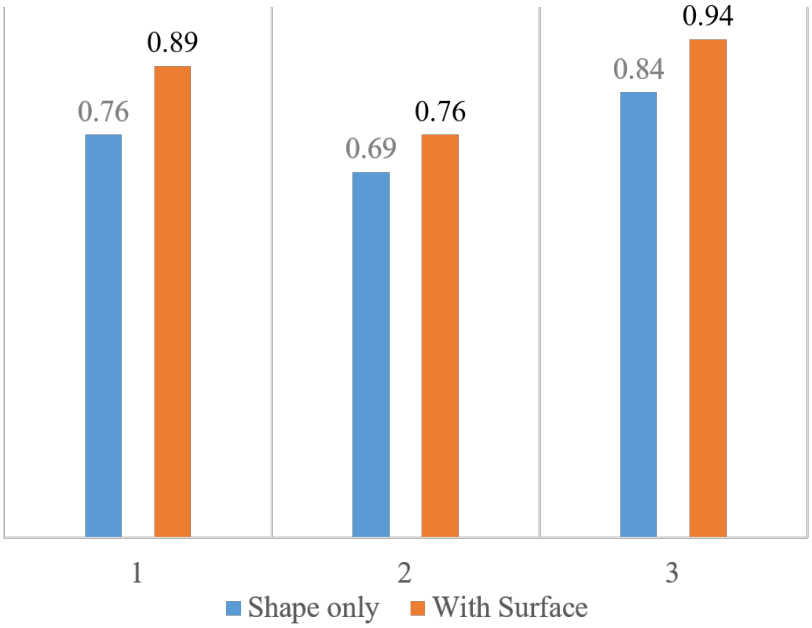

Fig. 3.14 The result with and without surface condition on 100 dataset.

As shown in Fig. 3.14, the blue bar is the prediction accuracy of these three objects without considering of the surface conditions, and the orange bar is the accuracy result of considering the surface conditions.

3.3 Summary

In this chapter, the surface condition based on vision, which is affecting the recognition rate, was able to be used as an information source of objects. However, due to the probability

distribution, the estimation range was restricted to only these three objects, and too much preprocessing was required in this method. Thus, the subsequent chapters focus on the different representations of PCD to reduce the processing.

Chapter 4

Touch Sensing Based Semi-Supervised Deep Learning

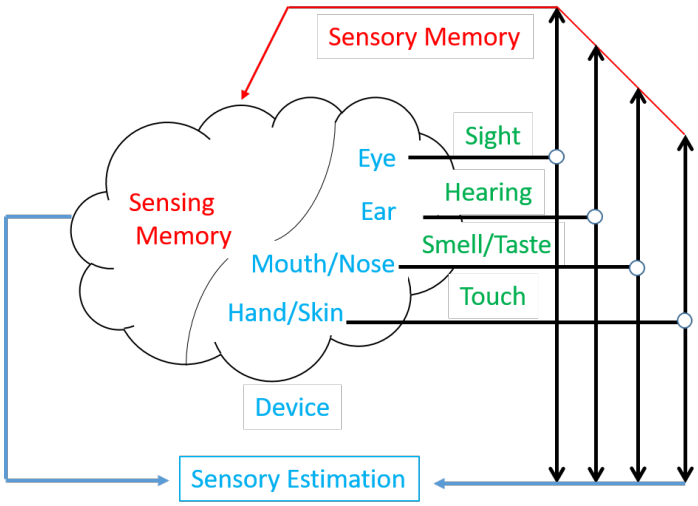

Fig. 4.1 Schematic illustration of integrating different sensors with memory devices for the mimicry of human sensory memory.

In the previous chapter, the PCD was analyzed and applied in the vision-based method, which considered the shape features and surface conditions of an object, and the results were shown as positive. But the truth is that the PCD is not captured by the RGB sensor, so that it does not correspond to the vision field. There are more positive potentials in PCD. As mentioned, the PCD is composed of points in space, which makes it fully relevant with geometrical features. But, the PCD includes the position value of points, which is only related to the reference coordinate. Once the reference is changed, then the previous relationship between these points is no longer available, i.e., one PCD is operated as a projection in a

different coordinate, and it outputs totally different results. In this section, a new approach is discussed to describe some stable features from PCD based on a human touch sensing theory.

As shown in Fig. 4.1, the human body has five sensation modalities, such as touch, sight, hearing, taste and smell, where these senses to the neural system have a specified priority. However, they are not isolated and often work together. These multiple sensing units with several memory areas build up a highly integrated system which is the human brain.

4.1 Touch Sensing Mechanism

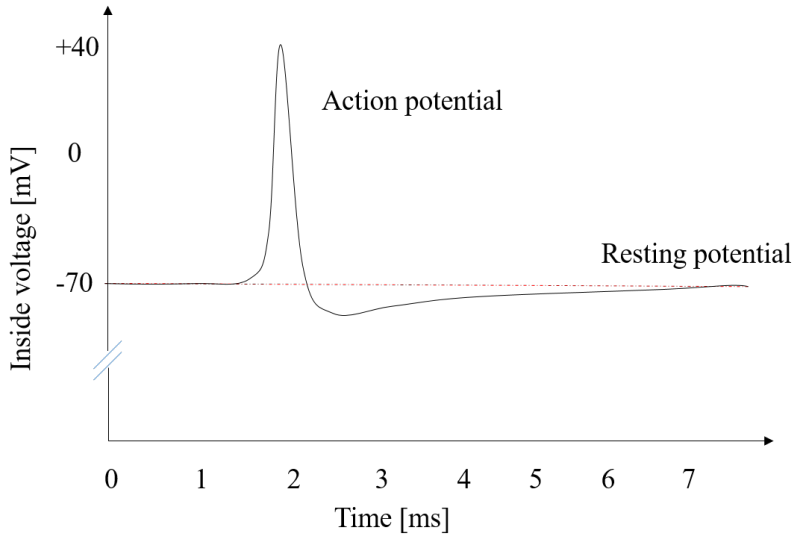

Fig. 4.2 A single action potential

Action potentials are often referred to as spikes or impulses [51, 52]. When it is not stimulated, a typical neuron rests at a voltage of about -70 mV. When a neuron cell is stimulated and it fires an action potential, the voltage jumps up to about $+40$ mV, and then comes back down again, as shown in Fig. 4.2. It all happens very fast (note the time scale), all in a matter of a few thousandths of a second. The action potential starts at a neuron's soma and travels all the way along its axon [53–55]. Basically, the shape of the action potential is the same anywhere along the length of the axon. In fact, the input signal is treated differently depending on the strength [56]. There are many types of strength that can be detect, because of the different types of neuron. Among them, the vibration stimulus and skin indentation are the main topics in this research.

More specifically, there are many different neurons under the skin such as the Pacinian corpuscle, Merkel cells, Meissner corpuscles, and Ruffini ending [57]. These sensing neurons

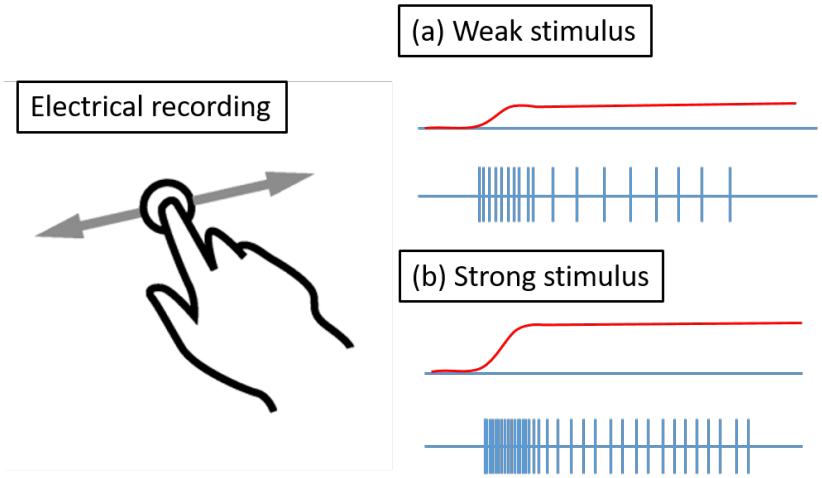

Fig. 4.3 The firing rate response to the pressure level

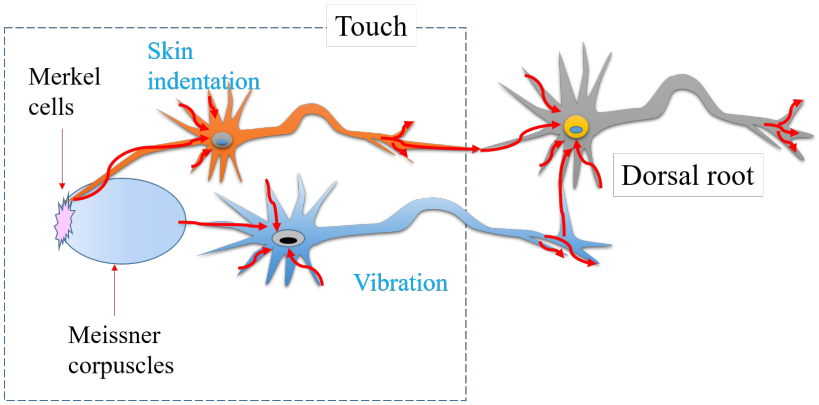

Fig. 4.4 A typical touch sensor neuron connecting via synapses

have different responses to different intensity stimuli, as shown in Fig. 4.3. These different morphologies serve as specific receptor functions, yet each is associated with a single neuronal cell type: the dorsal root ganglion neuron. Fig. 4.4 shows the neurological model of a typical touch sensor. In our touch sensing model, we only consider two main sensing neurons, the Merkel cells and the Meissner corpuscles. The Merkel cells correspond to the skin indentation, such as the pressure or the large changes of the surface. The Meissner corpuscles correspond to the vibration, e.g., the sense of surface roughness.

Stimulus to spike

A model for relating an input current to the spike probability is to characterize the average stimulus over a certain window of duration T preceding each spike. The collection of all stimuli that lead to spike generation is called the spike-triggered ensemble, and the mean of the spike-triggered ensemble is called the spike-triggered average (STA). If one considers the spike-triggered ensemble to represent a probability distribution over important stimuli, then the STA is an estimate of the first moment of that distribution. Intuitively, the shape of the spike-triggered average corresponds with the shape of the primary feature for which the neuron is selective. In shortly, a common model of a neuron is the idealization of a Poisson process, which produces the probability of the observed number y of spikes which is given by:

$$P(y|\lambda) = \frac{(\lambda\Delta)^y}{y^!} exp(-\lambda\Delta) \tag{4.1}$$

where $\lambda\Delta$ is the expected number of spikes in a small unit of time Δ, and λ is the intensity of the Poisson process. In practice, the binary representation of y simplifies the log-likelihood further

$$\ell(\theta) = \sum_{t=spike} (\log\lambda_t) - \Delta\sum_t(\lambda_t) \tag{4.2}$$

As shown in Fig. 4.5, on the basis of the Poisson process, a generalized linear model (GLM) of a neuron was proposed and applied by many neural scientists [52-55]. The equation of random spike in this GLM is as followed:

$$P(\cdot) = (S \in [t, t+\nabla]|H_t, X_t) = \lambda_t \tag{4.3}$$

where the $P(\cdot)$ represents the spike number between time t to $t+\Delta$; X is the stimulus in time t and the H is the neuron activation state. The outcome of current random spike generation is highly depended on the current X and post-spike activation H. In order to fully model the response, we need to identify the conditional intensity λ. The λ could be the outcome of the

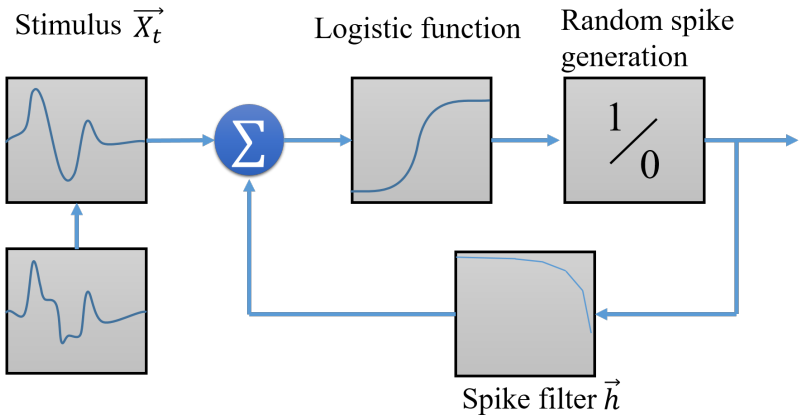

Fig. 4.5 The generalized linear model of a neuron

roll of a die, samples in Markov random field's integration period or the number of action potentials generated by neural network. It is shown in the following equation:

$$\lambda_t = f(k \cdot X_t + h \cdot H_t + \mu_i) \tag{4.4}$$

where the λ_t is related to the stimulus and previous spiking history, the k is the stimulus filter, h is a post-spike filter to account for spike history, and μ is a constant bias to match the firing rate. The term $(h \cdot H_t)$ is the post-spike activity received by the neuron. The f is an arbitrary invertible function termed a link function. Generally, the f is refereed as $exp(\cdot)$, which is able to simplify the likelihood calculation. Combining equations 4.1 and 4.4, the probability of observing the complete spike can be represented as:

$$\ell(\theta) = \sum_{i,t=spike} (k_i \cdot X_t + h_i \cdot H_t + \mu) - \Delta \sum_{i,t=spike} (k_i \cdot X_t + h_i \cdot H_t + \mu) \tag{4.5}$$

where we have added the subscript i throughout to label neuron i, and the θ is the parameter space. Up to here, obviously, this full procedure can be represented by a neural network with one input, and a recurrent layer as the post-spike generator. As shown in Fig. 4.6, a multiple GLM is used to simulate the neuron spike, where the input stimulus values are represented by the point position in the PCD. However, the (x,y) values are fixed in a PCD, so that only the z values are related to the generation of stimulus, and the z values are normalized. In order to apply this static data in the sequential model as stimulus inputs, the convolution is applied at the beginning layer to simulate the chronological order. By using the algorithm in Sec. 2.3.1, the sequential order n of a PCD can be calculated, in this case $n = 4096$, which is also referred as *spike* in equation. By applying this slight modified multiple GLM on a given

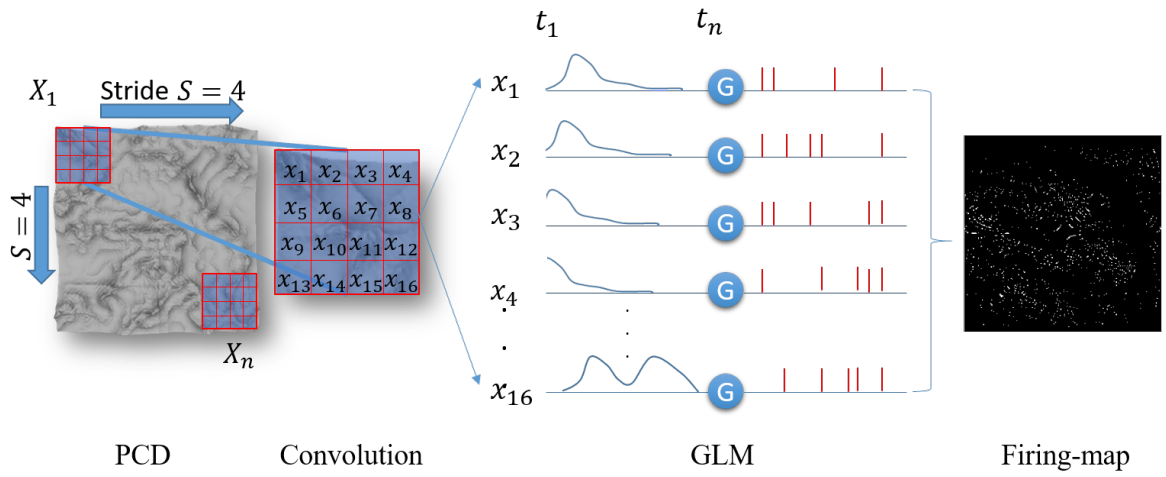

Fig. 4.6 Binary conversion of PCD to neuron spikes

PCD, a neuron firing map can be generated, which means how the roughness of this surface is felt when human slips the fingers over on it. By applying this less information, our neural network still is not able to converge.

4.2 Improved Vibration Firing Map

In the previous section, the experimental result of the multiple GLM on our PCD was not good enough. However, we can clearly see this theory is working. To be able to obtain more collectable features, we need more sensitive conditional intensity. To achieve this target, a recurrent structure is applied.

4.2.1 Stacked Convolutional Auto-encoder

Firstly, let us recall the basic principles of auto-encoder (AE) models. An AE takes an input $X \in \mathbb{R}$ and first maps it to the latent representation $h \in \mathbb{R}$ using a deterministic function of the type $h = f(\theta) = \sigma(Wx+b)$, where the parameter θ consist of W and b. These parameters are then used to reconstruct the input by a reverse mapping. The two parameter sets are usually constrained to be of the form $W = W^T$, using the same weights for encoding the input and decoding the latent representation. Each training pattern x_i is then mapped onto its code h_i and its reconstruction y_i. The parameters are optimized, minimizing an appropriate cost function over the training set $D_n = \{(x_0,t_0),...,(x_n,t_n)\}$. Fully connected AEs ignore the data structure. This is not only a problem when dealing with realistically sized inputs, but also introduces redundancy in the parameters, forcing each feature to be

global. Convolutional AEs (CAEs) differ from AEs because the weights in CAEs are not shared among all locations in the input, preserving spatial locality. The reconstruction is hence due to a linear combination of basic image patches based on the latent code [17–20].

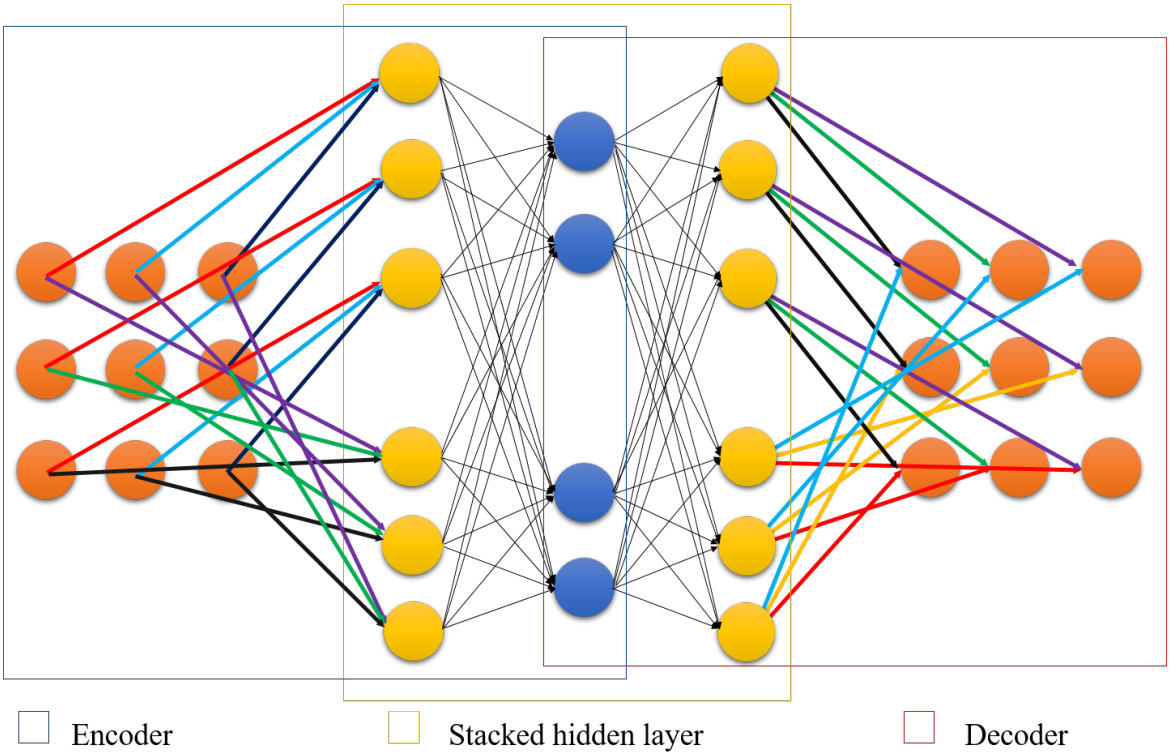

Fig. 4.7 Training structure of the autoencoder

The CAE architecture is intuitively similar to the CNN described in Sec. 2.3, except that the weights are shared during the encoding and decoding in each CAE, i.e., $\hat{W} = W^T$. And the structure is given by:

$$h^k = \sigma(x \cdot W^k + b^k) \tag{4.6}$$

where the bias is broadcasted to the whole map, σ is an activation function, and $\cdot$ denotes the 2D convolution. A single bias per a latent map is used, because a desired each filter is specialized on the features of the whole input. The reconstruction is obtained by using the following equation:

$$y = \sigma(\sum_{k \in H} h^k \cdot \hat{W}^k + c) \tag{4.7}$$

under the minimization of a cost function, which is described by the mean squared error (MSE):

$$E(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (x_i - y_i)^2 \tag{4.8}$$

where h identifies the group of latent feature maps and $\hat{W}$ is obtained by taking the flip operation over both dimensions of the weights W, i.e., $\hat{W} = W^T$. Just as for the standard networks the back-propagation algorithm is applied to compute the gradient of the error function with respect to the parameters. This can be easily obtained by convolution operations using the following formula:

$$\frac{\partial E(\theta)}{\partial W^k} = (x \cdot \delta h^k + \hat{h}^k \cdot \delta y) \tag{4.9}$$

where the δh and δy are the deltas of the hidden states and the reconstruction, respectively.

The AEs can be stacked to form a deep hierarchy, as shown in Fig. 4.7. Each layer receives its input from the latent representation of the previous layer. As for deep belief networks, unsupervised pre-training can be performed in greedy and layer-wise fashion. After wards the weights can be fine-tuned using back-propagation. The structure is very simple, as shown in Fig. 4.7, whose structure can be described as the following equations:

$$\alpha = (F_t \cdot X_t)\omega_t + b_t \tag{4.10}$$

$$\beta = S_1(\alpha)\omega_{s_1} + b_\alpha \tag{4.11}$$

$$\delta = S_2(\beta)\omega_{s_2} + b_\beta \tag{4.12}$$

$$y(\cdot) = S_3(\delta)\omega_{s_3} \tag{4.13}$$

where the α, β, and δ are the simply summaries of outputs from the previous layer, and S represents the activation function.

4.2.2 Long-short Term Memories Unit

According to the previous research, there is a memory effect in this GLM, which is defined as *Spike History*. To improve our CAE, a variant LSTM structure is applied between the hidden units. As shown in Fig. 4.12, it is the unfold structure of our proposed model. For general-purpose sequence modeling, LSTM as a special RNN structure has proven to be stable and powerful for modeling long-range dependencies in various previous studies [21, 29, 30, 84]. An LSTM has a more detailed and effective method to explain how memories affect on the current cell. The LSTM has the ability to remove or add information to the cell state, and the decision is made by structures called gates, i.e., forget gate, input gate, and output gate.

The gates are a way to optionally pass information through. They are composed out of a sigmoid layer and a pointwise multiplication operation. The sigmoid layer produces the numbers between zero and one, which describes how much of each component should be passed through.

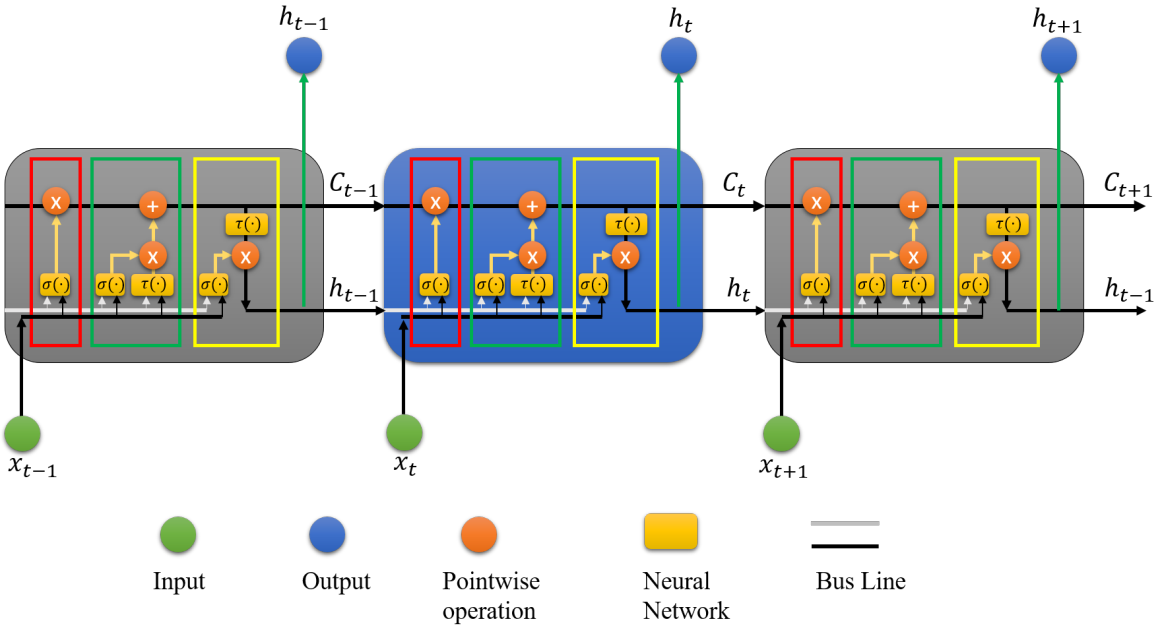

Fig. 4.8 Full structure of an LSTM

As shown in Fig. 4.8, the red block is called "*Forget gate*", which is the first and most powerful gate. It decides what information could be thrown from the current network cell. The gate's status is decided by a sigmoid layer, where it takes both h_{t-1} and x_t and produces a number between zero and one for each number in the cell state C_{t-1}. An output one represent *Completely keep* and 0 represents *Completely drop*. The mathematical representation of red block is:

$$f_t = \sigma(\omega_f^k \cdot h_{t-1} + \omega_f^p \cdot x_t + b_f) \tag{4.14}$$

where the ω represents the vector of weights, the subscription f represents the ω is in the forget gate, and the superscript k and p represent the ω respects to the previous hidden cell and current neuron input, respectively. The next step is to decide what new information can go to be stored in the cell state, as shown in green block of the Fig. 4.8. This gate consists of two parts: one is a sigmoid layer called *Input gate*, which decides what values should be updated, and a tanh layer creates a vector of new candidate values. The other is composed of a new candidate $\hat{C}_t$ and the output of *Input gate*, which are combined to create an update to

the cell state. The mathematical representation of green block is:

$$i_t = \sigma(\omega_i^k \cdot h_{t-1} + \omega_i^p \cdot x_t + b_i) \tag{4.15}$$

where the the subscription i of ω respects to the input gate.

$$\hat{C}_t = \tau(\omega_C^k \cdot h_{t-1} + \omega_C^p \cdot x_t + b_C) \tag{4.16}$$

where the the subscription C of ω respects to the cell status, and τ is the tanh logistic function. The output of tanh function is permitted to assume both positive and negative values in the interval $[-1,1]$, that allows the function performs as a amplifier. Therefore, the new cell status C_t can be updated as:

$$C_t = C_{t-1} \times f_t + (i_t \times \hat{C}_t) \tag{4.17}$$

The values of the output gates are computed together with the new state of the memory unit, and their outputs are calculated as follows:

$$o_t = \sigma(\omega_o^k \cdot h_{t-1} + \omega_o^p \cdot x_t + b_o) \tag{4.18}$$

$$h_t = o_t \times \tau(C_t) \tag{4.19}$$

4.2.3 Short Term Memories CAE

According to the LSTM and the GLM, a specified NN with memory units is designed, where the AE is the kernel trainer and a convolution layer is the input layer. Table 4.1 is the structure of a stacked CAE (sCAE), which is used to convert the PCD to a brain fire-map.

Table 4.1 The specified structure of a sCAE, where the numbers in output shape represent width, height and depth of output layer

Layer	**Output Shape** Width, Height, Depth	**Parameters**	**Connection**
Input	26, 32, 1	0	
Conv. 1	26, 32, 16	13312	Input Layer
Maxpooling	13, 16, 32	0	Conv. 1
Conv. 2	13, 16, 8	1664	Maxpooling
Upsampling	26, 32, 32		Conv. 2
Conv. 3	26, 32, 1	832	Upsampling

There is an improved model of memory cell, as shown in Fig. 4.9. In this model, the cell's status at $t-1$ cell no longer affects cell at $t+1$. The reason is that vibration or skin

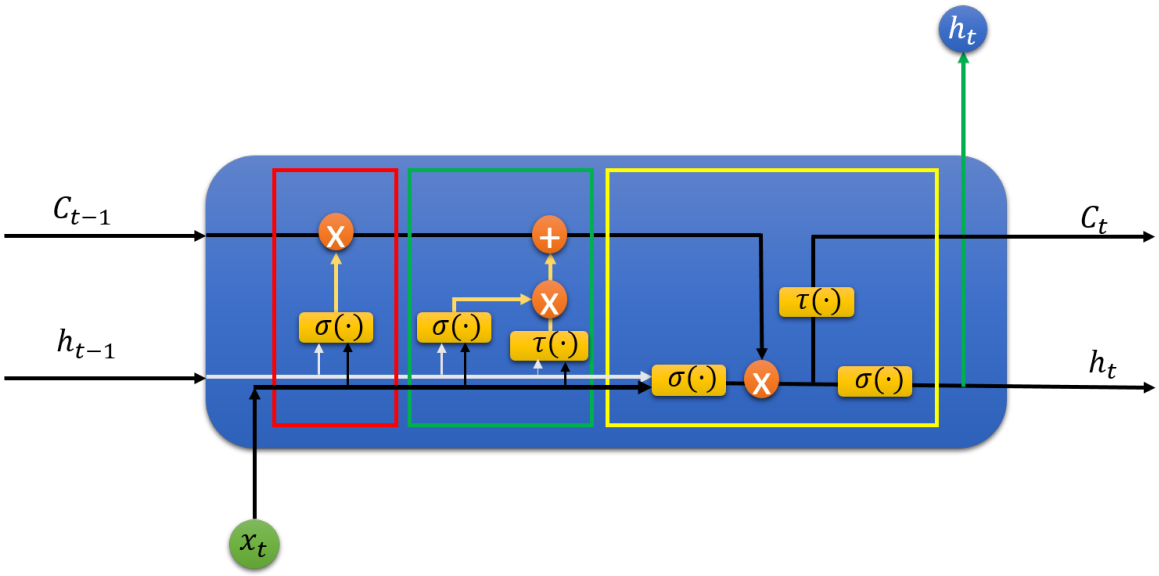

Fig. 4.9 A variant model of an LSTM

indentation spikes in only firing in very short time, and reverts back to the rejection mode immediately. The details of this unit are shown in Fig. 4.10, and the complete structure of sMCAE, which consists of this unit is shown in the figure Fig. 4.11. The equations of LSTM are replaced according to the new proposed model as follow:

$$f_t = \sigma(\omega_f^k \cdot h_{t-1} + \omega_f^p \cdot x_t + b_f) \tag{4.20}$$

$$\hat{C}_t^1 = C_{t-1} \times f_t \tag{4.21}$$

$$A_t = \tau(\omega_a^k \cdot h_{t-1} + \omega_a^p \cdot x_t + b_a) \tag{4.22}$$

where the subscription a represents the amplifier gate.

$$i_t = \sigma(\omega_i^k \cdot h_{t-1} + \omega_i^p \cdot x_t + b_i) \tag{4.23}$$

$$\hat{C}_t^2 = A_t \times i_t \tag{4.24}$$

$$T_t = \hat{C}_t^1 + \hat{C}_t^2 \tag{4.25}$$

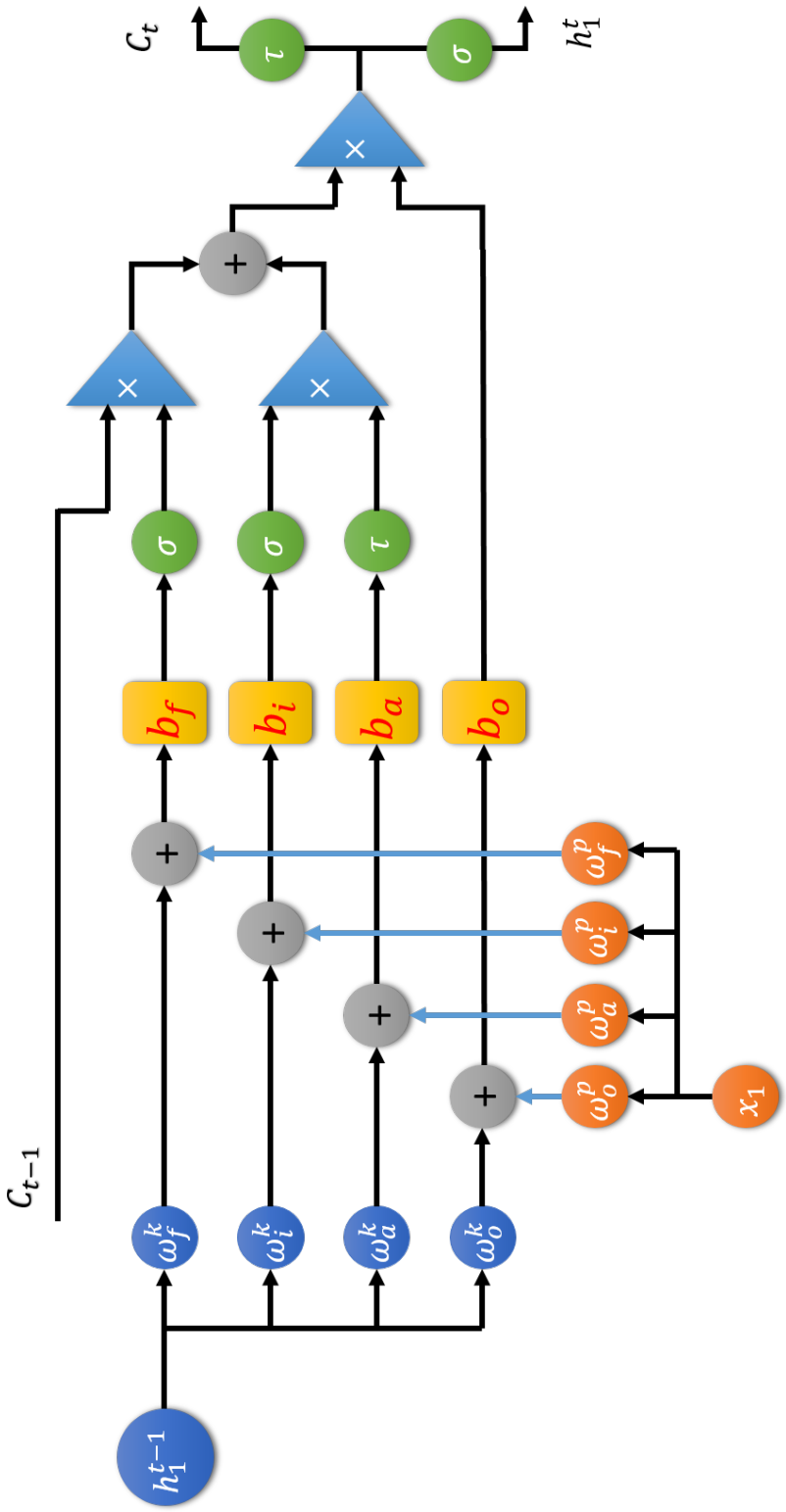

Fig. 4.10 Illustrated detail of a short-term memorise unit

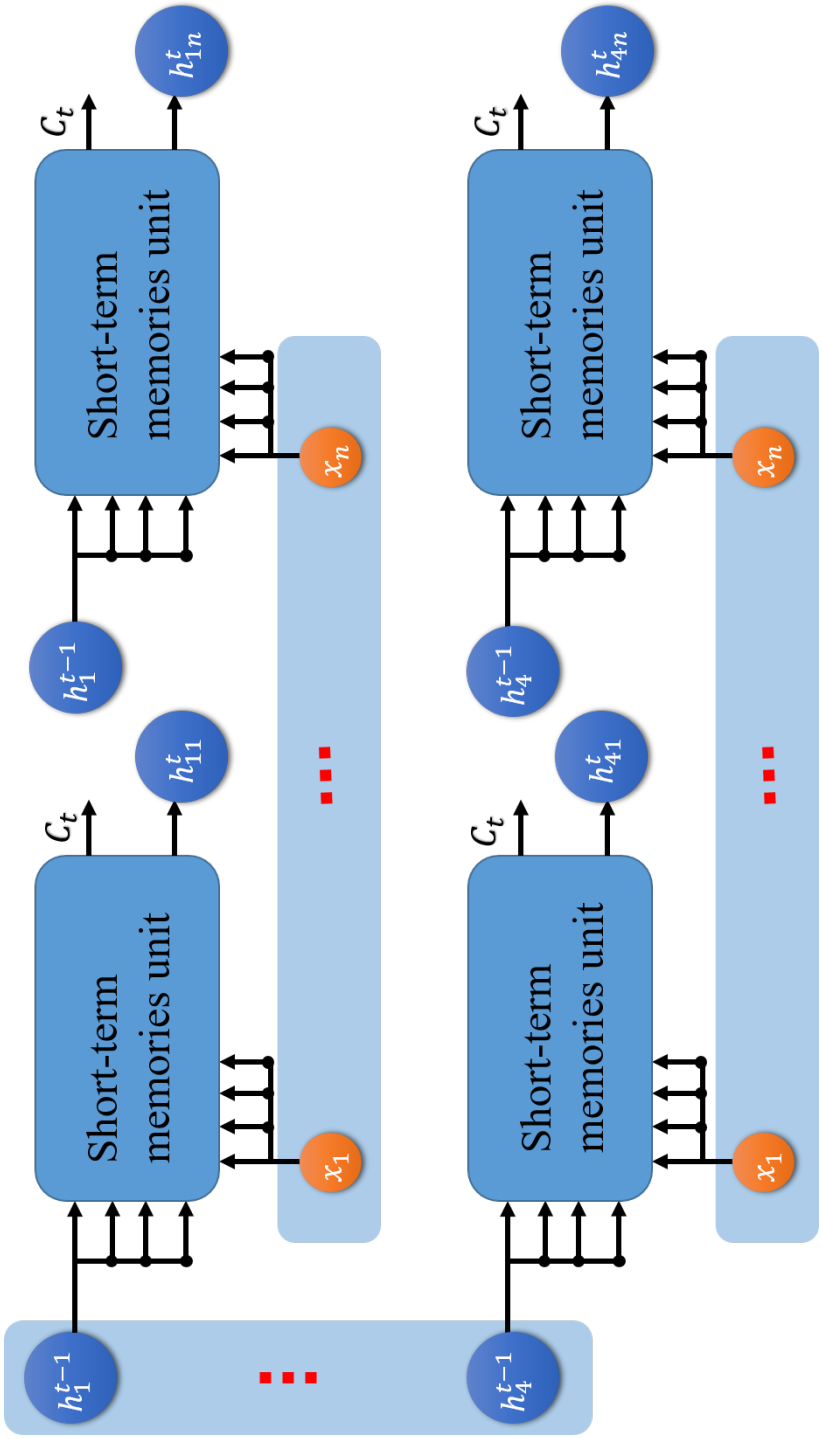

Fig. 4.11 Full structure of our model

where the T represents the current inner cell status. Then the passing status and output value are shown as:

$$C_t = \tau(\hat{T}_t \times x_t + \hat{T}_t \times h_{t-1}) \tag{4.26}$$

$$h_t = (\hat{T}_t \times x_t + \hat{T}_t \times h_{t-1}) \tag{4.27}$$

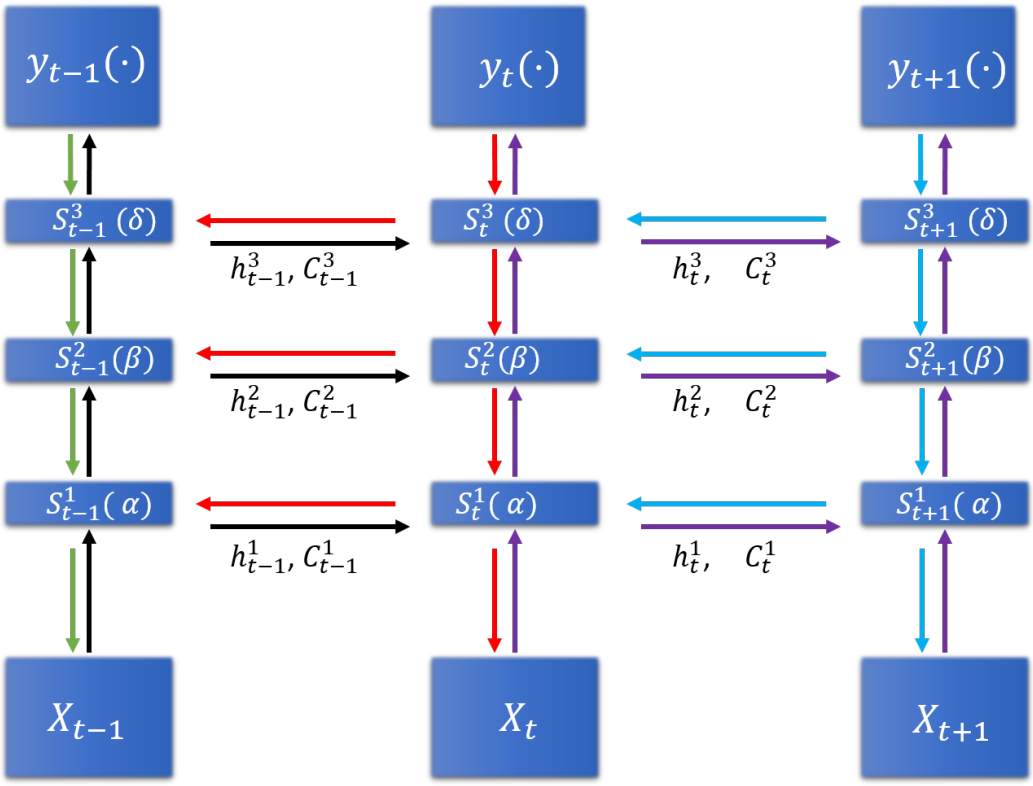

Fig. 4.12 The unfold structure of the sMCAE.

However, the learning algorithm of this short-term memories CAEs (sMCAE) is also a little bit different from an original back-propagation. Since a short memory unit is only activated one step forward, the training also should be calculated one step backward, which are shown in Fig. 4.12 as the same color arrows, i.e., the red arrows show that the back-propagation between $y_t(\cdot)$ and X_t affects also the previous weights in $t-1$. The forward procedures are shown as the following equations:

$$\begin{aligned} y_t &= f(S^3_t) \\ S^3_t &= f(S^2_t, C^3_{t-1}, h^3_{t-1}) \\ S^2_t &= f(S^1_t, C^2_{t-1}, h^2_{t-1}) \\ S^1_t &= f(X_t, C^1_{t-1}, h^1_{t-1}) \end{aligned} \tag{4.28}$$

Then, the short memories back-propagation is given by:

$$E_t = \frac{1}{2n}\sum_t (y_t - x_t)^2 \tag{4.29}$$

$$\frac{\partial E_t}{\partial \omega_t} = \frac{\partial E_t}{\partial S_t^3}\frac{\partial S_t^3}{\partial S_t^2}\frac{\partial S_t^2}{\partial S_t^1}\frac{\partial S_t^1}{\partial \omega_t} \tag{4.30}$$

Up to here, it is nothing more different than a normal back-propagation. However the previous weight ω_{t-1} also takes an effect from the C_{t-1} and h_{t-1}. The S is the short memories block, and the ω_i^j represents the four weights in time i of jth short memories block. To the ω_{t-1}, the E_{t-1} gives another back propagation training.

$$g(x) = \eta \cdot f\left(\frac{\partial \sum_{m=3}^{i}\sum_{m=3}^{j}\omega_{t-1}^i x_t \omega_{t-1}^j x_t}{\partial \omega_{t-1}^{old}}\right) \tag{4.31}$$

where $f()$ represents the ReLU function, and η is the learning rate. Since the ReLU is defined as$f(x) = \max(0,x)$, then the dervative of ReLU is:

$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.32}$$

Finally to a new ω_{t-1}, it follows that:

$$\omega_{t-1}^{new} = \omega_{t-1}^{old} - g(\cdot) \tag{4.33}$$

As an experiment, the NN kernel which is composed of ω is set to 100 and 9 for the first and second encoder layers, which can help us to evaluate the fitness of the kernel. The trained kernels after 2000 epochs on 900 groups are shown in Fig. 4.13 (a) and (b). As shown in Fig. 4.14, the PCD is converted to the neuron fire-map. The labels K_1 and K_2 are the two slightly changed structures of this memorable sMCAE: the result K_1 shows the convert result only with one recurrent stage, whereas K_2 shows the result with both recurrent stages.

4.3 Skin Indentation

As mentioned at the beginning of this chapter, to build a robust NN, the more information is required. In the previous section, the sMCAE were used to convert a PCD into the brain firing map of vibration signal. But, only vibration is not enough. Thus, the skin indentation

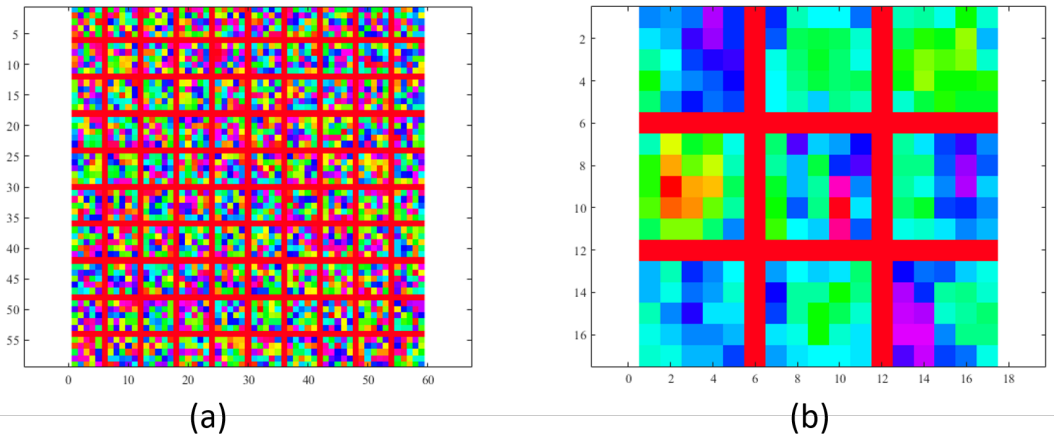

Fig. 4.13 Visualized maps of hidden layers: (a) denotes the visualized maps of first hidden layer's 100 kernels and (b) is the visualized maps of second hidden layer's 9 kernels

is discussed. To manage that, a set of simple definitions of the surface are defined. In this research, any complex surface is assumed to be composed of several simple structures: i.e., they are defined as a curved plane, a normal plane, and an inclined plane, as shown in Fig. 4.15. However, these definitions need to be more measurable. In many previous research, it is proved that the surface normal vector is the most intuitive variables to the surface. In this section a new scheme is proposed to quantify the difference of surface curvatures.

4.3.1 Data

Table 4.2 Parameters for the captured data

Captured area	**X direction**		**Y direction**		**Z direction**	
	Positive	*Negative*	*Positive*	*Negative*	*Positive*	*Negative*
	75 mm	75 mm	50 mm	50 mm	650 mm	0 mm
Object size	Half quoter cylinder Length × Width × Height		Triangular prism Length × Width × Height		Trapezoidal Length × Width × Height	
	80 mm×50 mm×16 mm		60 mm × 56 mm × 18 mm		100 mm × 40 mm× 22 mm	
The average number of points	8981		8978		8987	

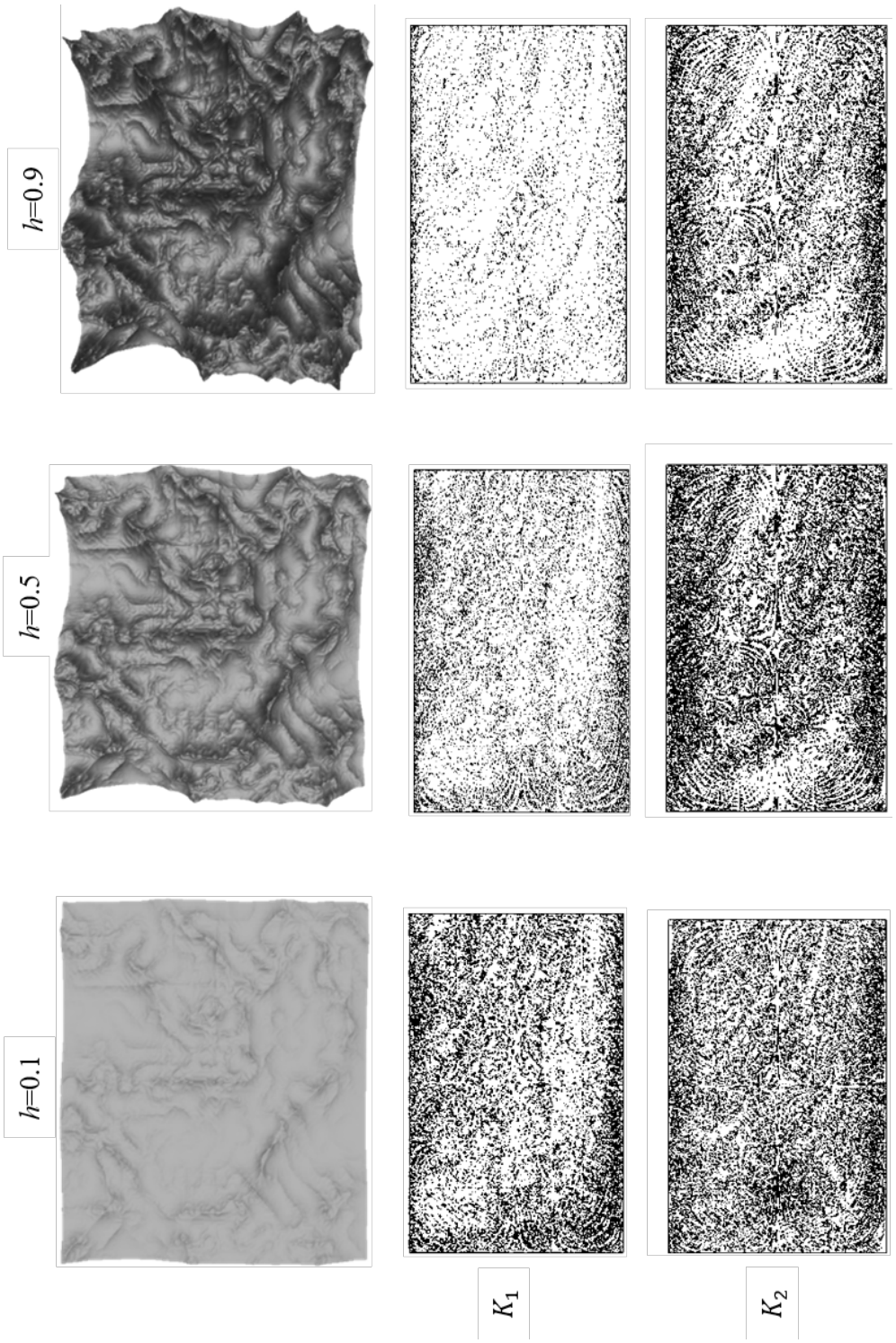

Fig. 4.14 The firing maps obtained from PCD

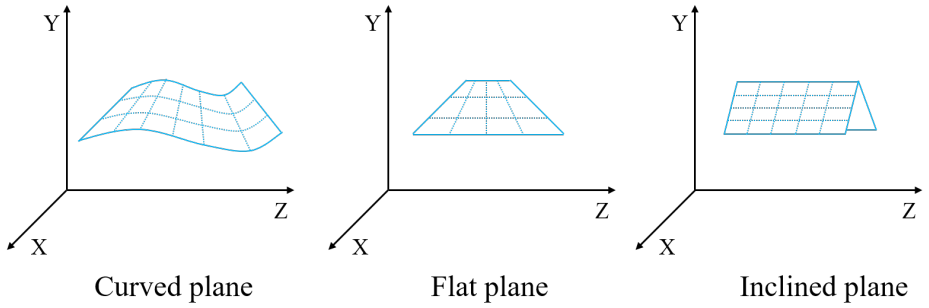

Fig. 4.15 The definition of main planes

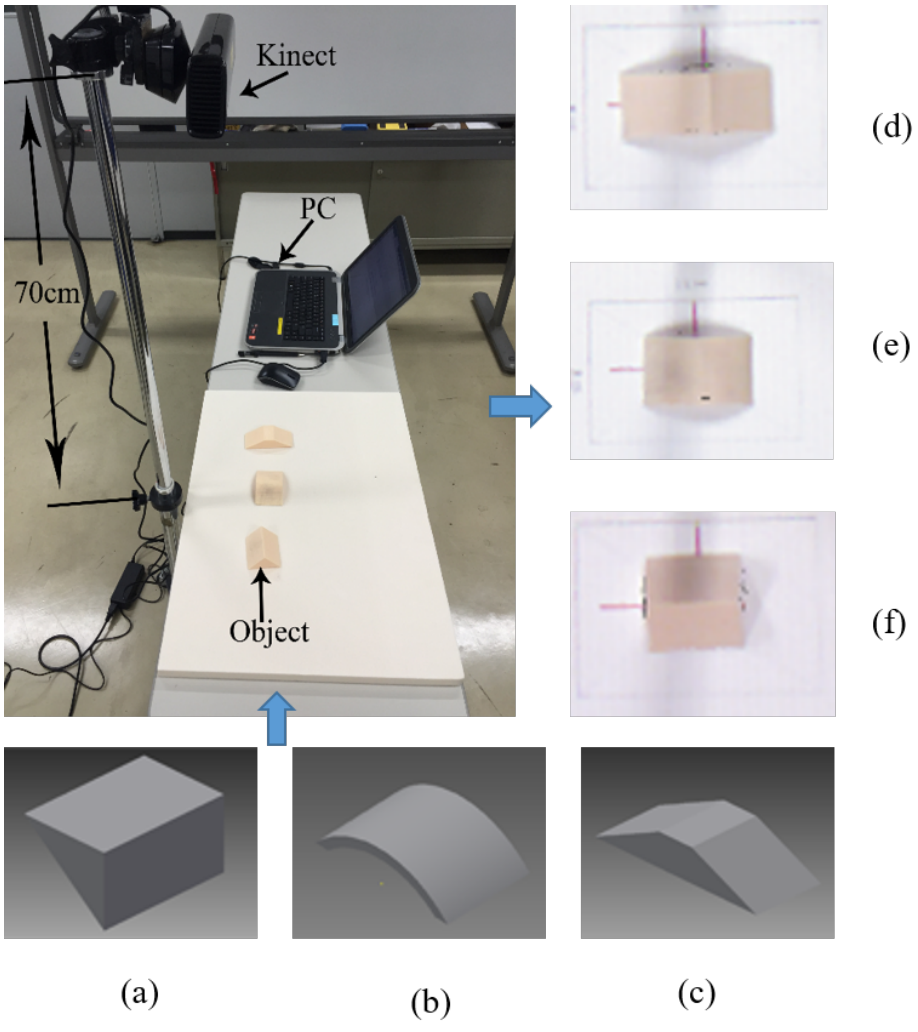

Fig. 4.16 Overview of the experiential environment: (a) CAD model of Triangular; (b) CAD model of Half quoter cylinder; (c) CAD model of Trapezoidal prism; (d) Depth data of the Trapezoidal prism; (e) Depth data of the Half quoter cylinder; and (f) Depth data of the Triangular

The environment that obtained the data is shown in Fig. 4.16. In order to obtain a set of quality point data, a fixed position is applied between the Kinect sensor and the object. This distance is determined by calculating the density of PCD in a fixed area. The definition of the fixed area is also shown in Table 4.2 as a captured area. Objects used to capture the data are 3D printed from CAD models to control the amount of captured data. All these works help us to save a lot of time on pre-processing of the point data. The full parameters are shown in Table 4.2, where each object has obtained the fifty-group data about 9000 points in each group.

4.3.2 Surface Condition

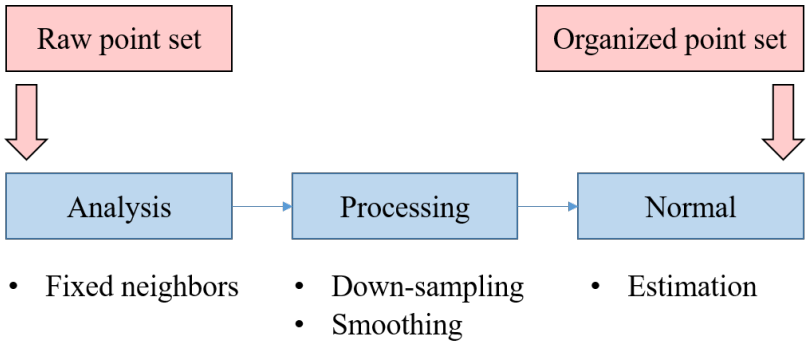

Fig. 4.17 Processing chart of point cloud data

The raw data captured from Kinect include a huge number of noise data. It cannot be directly applied in research. The raw point set is converted to the organized point set by down-sampling operation and smoothing, as shown in Fig 4.17. The organized point set can be used on a surface normal estimation [41–50].

Assume that the normal vectors v of a curve line on each point P are taken as unique pressure forces N. Therefore each pressure force N has a maximum force value which does not exceed one unit of pressure. As shown in Fig. 4.18, the force vector m can be separated into two branch forces in 2-dimensional space, the n_x and n_y. By using the n_x and n_y, it is easy to describe the change of this line's curvature. Once it is turned into 3-dimensional, there is n_z as the branch force in Z-axis, as shown in Fig. 4.19. The mathematical equations for this theory are shown such as:

$$m = n_{\mathrm{x}} e_{\mathrm{x}} + n_{\mathrm{y}} e_{\mathrm{y}} + n_{\mathrm{z}} e_{\mathrm{z}} \tag{4.34}$$

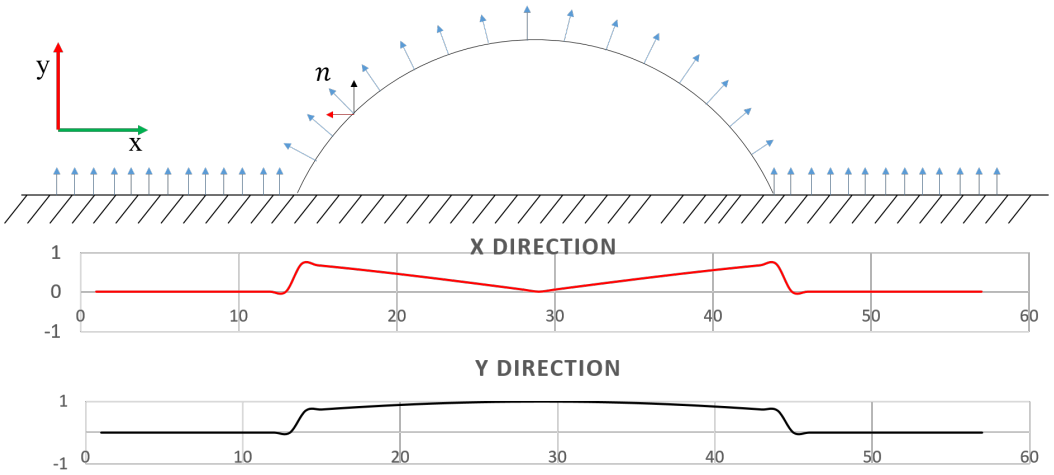

Fig. 4.18 The concept of surface-common-feature

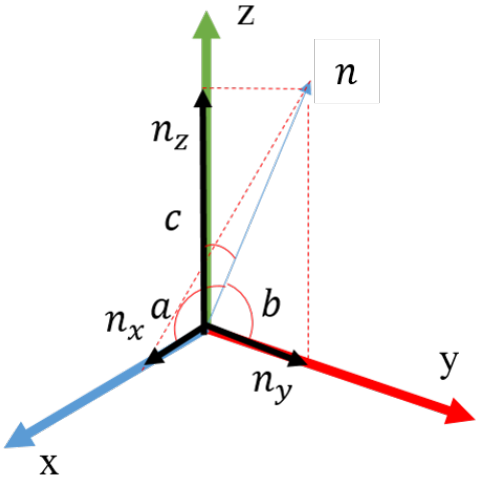

Fig. 4.19 Direction cosines of a normal vectors

where e_x, e_y, and e_z are the standard bases in Cartesian notation, then the direction cosines are

$$\cos a = \frac{m \cdot e_x}{|m|} = \frac{n_x}{\sqrt{n_x^2 + n_y^2 + n_z^2}} \tag{4.35}$$

$$\cos b = \frac{m \cdot e_y}{|m|} = \frac{n_y}{\sqrt{n_x^2 + n_y^2 + n_z^2}} \tag{4.36}$$

$$\cos c = \frac{m \cdot e_z}{|m|} = \frac{n_z}{\sqrt{n_x^2 + n_y^2 + n_z^2}} \tag{4.37}$$

Squaring each equation and adding the results, it follows that:

$$\cos^2 a + \cos^2 b + \cos^2 c = 1 \tag{4.38}$$

After obtained the angles a, b and c, the branch force of n can be calculated as:

$$\begin{aligned} n_x &= \hat{n} \cos a \\ n_y &= \hat{n} \cos b \\ n_z &= \hat{n} \cos c \end{aligned} \tag{4.39}$$

where $\hat{n} = \sqrt{n_x^2 + n_y^2 + n_z^2}$.

By using this method, the surface-condition-feature (SCF) map can be converted from the surface normals. An X-Z view of a half quoter cylinder's SCF map is shown in Fig. 4.20, where the positive part of this line represents the *Upper curve*, and the negative part represent the *Downward curve* (See as red and black lines in Fig. 4.20). And it is very easy to remove the noise data and corner plane data, since their normal vector is perpendicular to the reference plane (See as green block in Fig. 4.20).

4.3.3 Result

When the full PCD is converted, a surface condition can be represented by all the cells of two small adjacent area's feature elements: $\{|D_i|\}_i^n$ and $\{|b_i|\}_i^n$. An encoder are used to convert these definitions into a skin indentation signal. Thus, the different skin indentation is roughly separated into six class, such as "Upward inclined," "Downward inclined," "Upward curve," "Downward curve," "Edge," and "Flat."

An sAE is used as a training algorithm in this part, as shown in Fig. 4.21. In order to use the autoencoder, the input data is reshaped into 18 different groups with 3×3 matrix data in

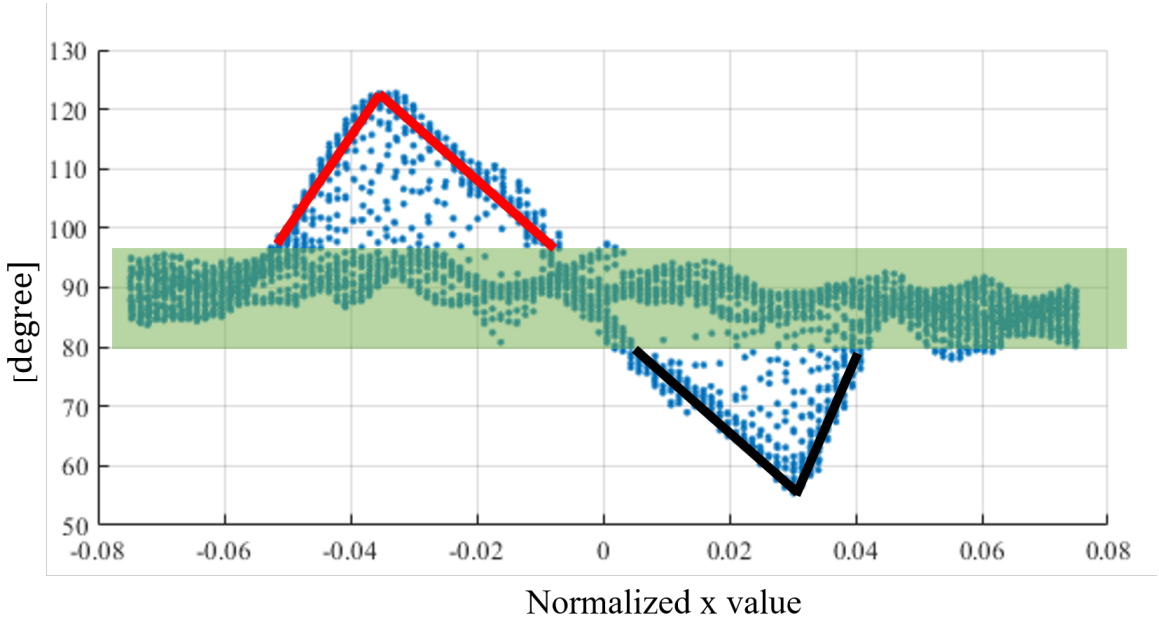

Fig. 4.20 Assumed skin indentation of half quoter cylinder in X-Z views

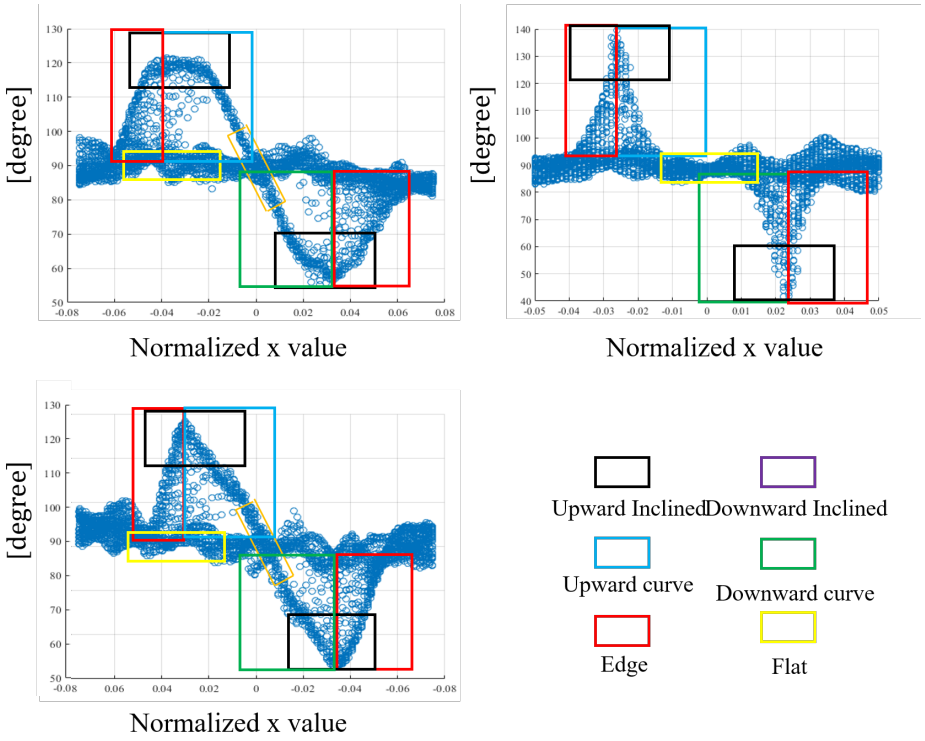

Fig. 4.21 (a) Input data from trapezoidal prism; (b) Input data from triangular prism; (c) Input data from half quoter cylinder

each group by considering the geometric features in the real world, such as: upward inclined (black), downward inclined (purple), upward curved (blue), downward curved (green), edge (red) and flat (yellow). Although, the data still include many noises, but the geometric features are strong enough to use average pooling to initialize the feature map.

4.4 Surface Common Feature

So far, the vibration firing map and the skin indentation are captured from a single PCD. However, during the process, the computing load for the skin indentation was huge, and the sMCAE computed the small curvature of the surface. For this reason, an improvement of surface common features (SCF) are proposed to only focus on the big changes of the surface. The method for modeling the skin indentation is based on the surface normal vectors [37, 39, 40]. A surface can be expressed by the normal vector n. In particular, two adjacent normal vectors are formed to express a minimum surface condition by calculating the distance vector $\overrightarrow{D}$. The processing of point cloud data with n points is calculated as the following steps:

1. Calculate all point vectors $\overrightarrow{P}_n$. The point vector is started from the origin point O to each point P_i, where $\{i \in n\}$.
2. Each of two adjacent point vectors $\overrightarrow{P}$'s are used to generate a tangent vector $\overrightarrow{T}$.
3. A normal vector $\hat{n}$ is considered as the cross product of two adjacent tangent vectors $\overrightarrow{T}_i$ and $\overrightarrow{T}_{i+1}$.
4. The difference between each two adjacent normal vectors are defined as the distance vector $\overrightarrow{D}$, which can represent the bending level between the two estimated minimum surfaces by its size $\|\overrightarrow{D}\|$.
5. The direction parameter b is assigned by 1 or 0, the values are dependent on the positive or negative direction of distance vector $\overrightarrow{D}$.

In this way, an environmental irrelevant bending features can be captured easily, as shown in Fig. 4.22.

4.5 Summary

In this chapter, an improved method which was based on SCF has been proposed under the human touch sensing theory. This method used 3D PCD in a specified deep learning structure,

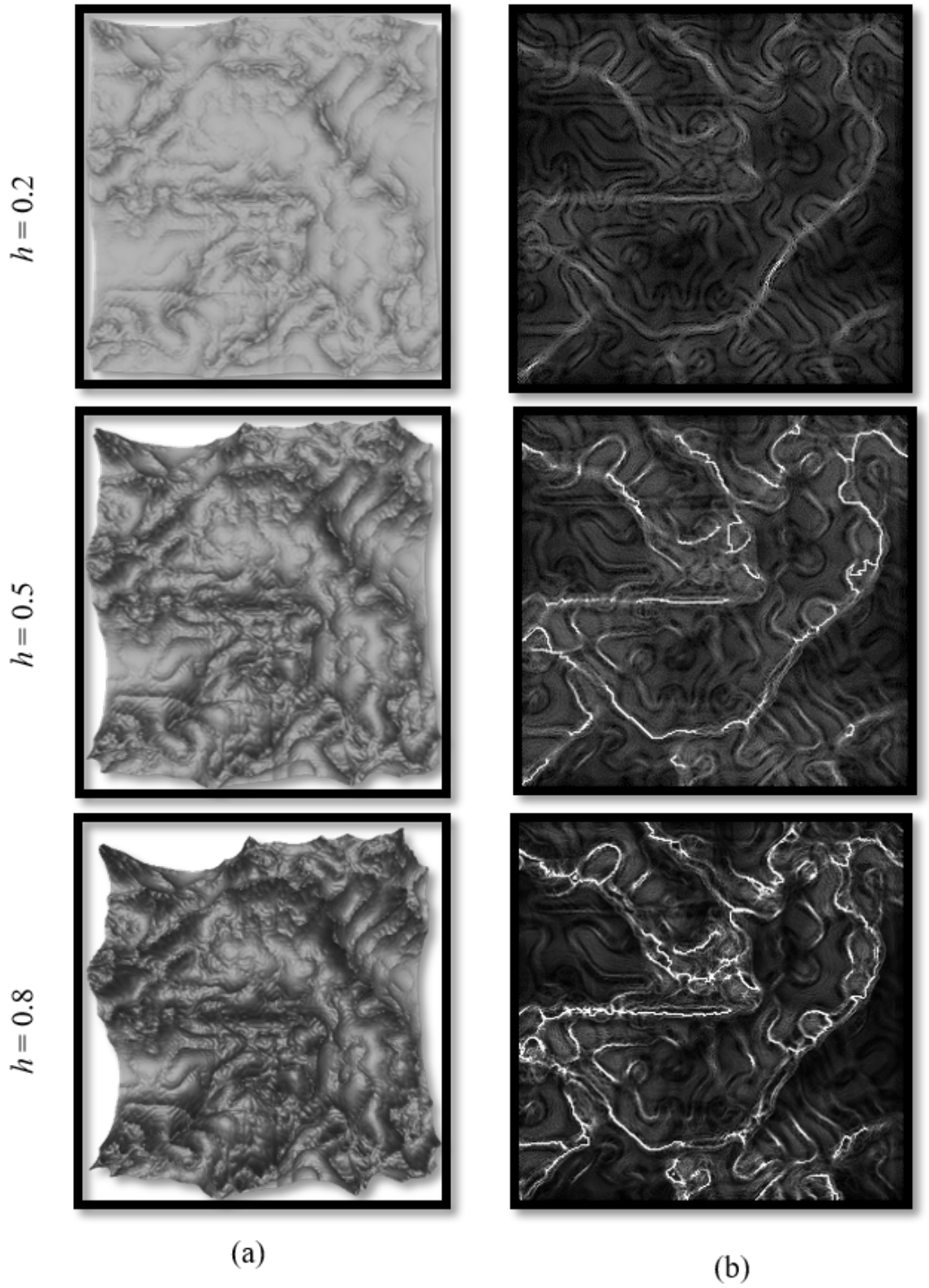

Fig. 4.22 The improved surface condition

i.e., the sMCAE, to produce the surface features. As expected, the result showed the SCF was very sensitive to the change of surface, and using SCF in sMCAE was very effective on the surface classification. However, the method was only proved by the well-formed idealized data. The next chapter will present this method on the captured real-world data, to improve the universality.

Chapter 5

Applications of Surface Common Features on Deep Learning

5.1 Facial Expression Recognition

5.1.1 Introduction

The facial expressions are response to internal emotional states, intentions, or social communications of humans [53-55]. Recognizing and understanding the facial expressions are of increasing importance because they provide the emotions, health status and affection of humans [56]. Such information can be used for controlling the interaction in HRI. For instance, analyzing the human emotions through these information, an intelligent system is able to support the user by offering help [57].

Over the last few years, many research have been proposed to recognize the facial expressions. Ying-li et al. [57-59] proposed a facial expression analysis system based on both permanent facial features and transient facial features, and they achieved an average recognition rate of 96 percent. Some other approaches employ geometry and appearance based features that are directly extracted from the face. For examples, Bartlett et al. [60] used a bank of 72 complex-valued Gabor filters, in which eight orientations and nine spatial frequencies were used to extract features and afterwards to detect the action units using support vector machine classifiers.

However, human emotions are complicated, thus the emotional expressions do not easily give the interpretation and inference of interaction. Since some emotional facial expressions have a very similar physical performance in 2D image data source, to distinguish a difference between sad and anger in images, it could be a difficult task even for human. In this section, the previously presented model is used in a recognition of emotional facial expression task.

5.1.2 Emotion Modeling

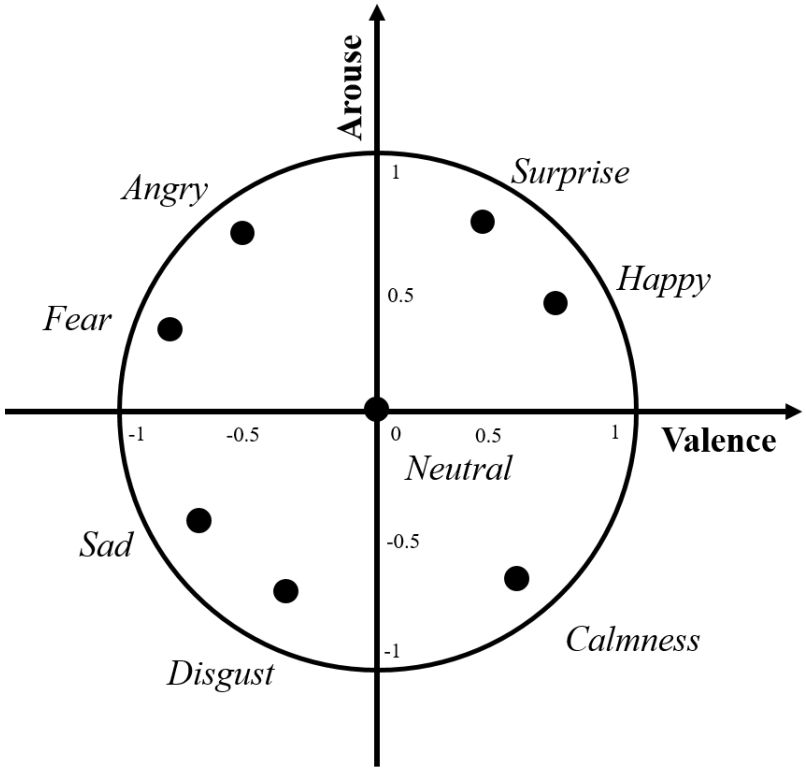

Fig. 5.1 A modified valence-arousal space

The valence-arousal space (VAS) is used to classify the emotions by the valued labels, where the simplified version is shown in Fig. 5.1. This map discrete is created based on the psychological model of Russell's circumplex model [62, 63]. This value labels of the classification model can lead to a robust emotional state representation that is continuous in principle. However, this model only can help us to quantify the emotion in two values. To obtain labels for these emotions, we collected some linguistic descriptions [61, 64-68]:

- **Happiness:** A face with happiness has corners of the mouth drawn up and back. The cheeks are raised, and the lower eyelids are tense. Wrinkles exist at the outer corners of the eyes.
- **Sadness:** A face showing sadness has the eyebrows drawn in and up, the skin below the eyebrows is triangulated with the inner corner up, and the corners of the lips are turned down. The jaw comes up and the bottom lip pouts out.

Table 5.1 Encoded labels based on VAS, where for each arousal or valence, the left denotes the high level flag; middle the neutral level flag; and right the low level flag

Emotions	**Eyes**						**Cheek**						**Mouth corner**					
	Arousal			**Valence**			**Arousal**			**Valence**			**Arousal**			**Valence**		
Angry	1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0	1
Sad	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1
Fear	1	0	0	0	1	0	0	0	0	0	0	1	1	0	0	0	1	0
Calmness	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0
Happy	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	1	0	0
Surprise	1	0	0	1	0	0	1	0	0	0	1	1	1	0	0	0	1	0

- **Disgust:** A disgusted face has the eyebrows downcast. The lower eyelid raised, the cheeks are raised and the nose is scrunched. The upper lip is also raised or curled upward.

- **Surprise:** A surprised face features the eyebrows raised up and curved. The skin below the brow is stretched. The eyelids are wide open. The jaw is dropped, widly opened mouth.

- **Fear:** A face showing fear has raised eyebrows that are usually more flat, not curved. The upper eyelids are raised, but lower eyelids are tense and drawn up. The lips are usually tensed or drawn back, the mouth may be open and nostrils may be flared

- **Anger:** An angry face will show eyebrows that are lowered and drawn together, widly opened eyes, the lower eyelids tensed. Nostrils may be flared, and the mouth is either firmly pressed together with the lips drawn down at the corners, or in a square shape as if shouting. Also, the lower jaw juts out.

Such as the literature descriptions of eyes in emotion, “Angry” is “bulging,” then the high level performance flag of “Arousal” is assigned as 1, whereas for the eye line in emotion, “Angry” is also described as ”lowered,” then the low level performance flag of the “Valence” is assigned as 1. Therefore the “angry” is encoded to {1,0,0,0,0,1}, and others so on. In fact, this encoding system allows us to separate the performance values more than 10 levels, but as a simple task in this case, it is not necessary. To predict an emotion, the VAS values of six parts are fed to a back-propagation neural network. And the labels of the emotions are encoded accoring to the descriptions, such as “Angry” = {1, 0, 0, 0, 0, 0 }; “Sad” = {0, 1, 0, 0, 0, 0 }; “Fear” = {0, 0, 1, 0, 0, 0 }; “Calmness” = {0, 0, 0, 1, 0, 0 }; “Happy” = {0, 0, 0, 0, 1, 0 }; and “Surprise” = {0, 0, 0, 0, 0, 1 }. In this network, we have 36 neurons in the input layer, and 18 neurons in the first hidden layer and 6 neurons in the output layer.

5.1.3 Data Training

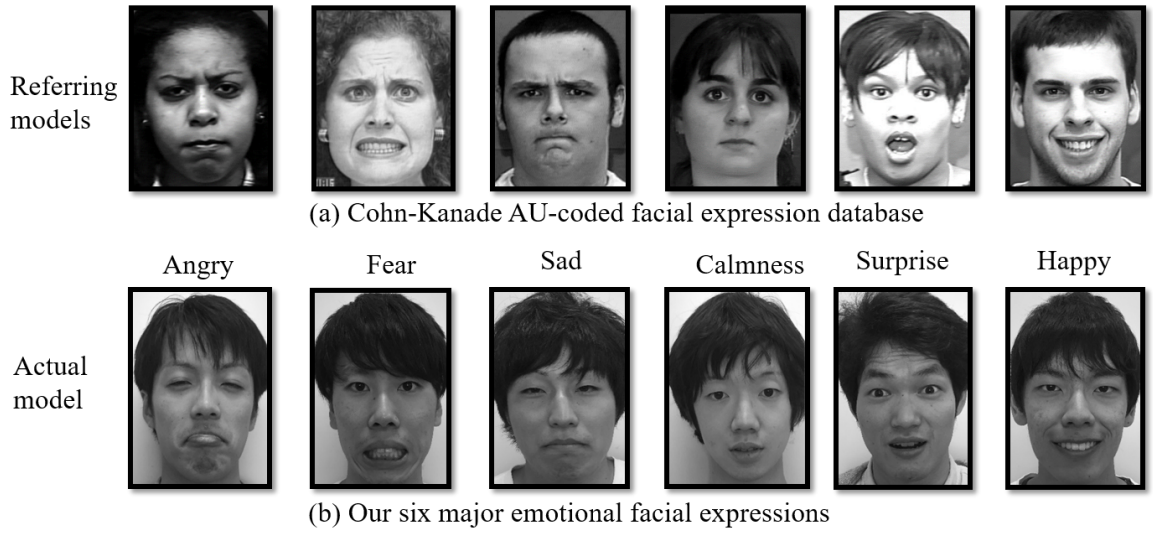

Fig. 5.2 The referred database: (a) Cohn-Kanade AU-coded facial expression database and (b) our six major emotional facial expressions, where they were captured in 230 frames from 30 actors.

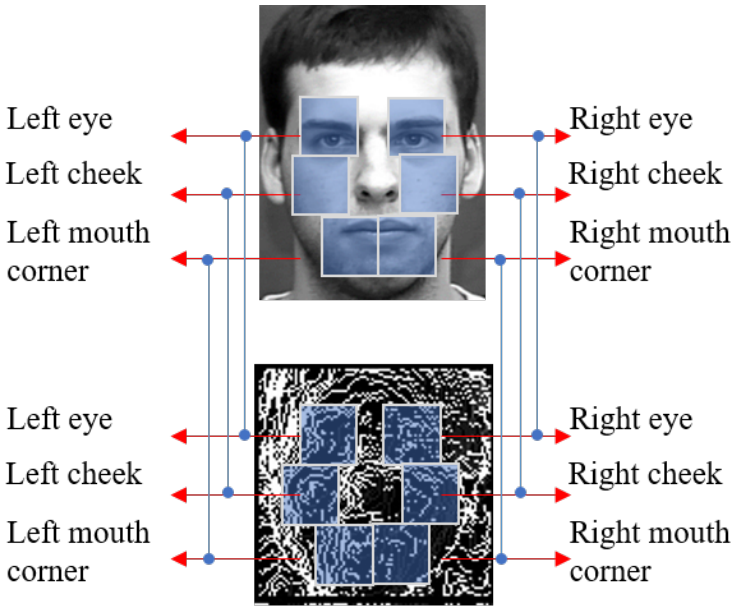

Fig. 5.3 The definition of six facial parts

As shown in Fig. 5.2, we referred Cohn-Kanade [59] facial expression database as our teaching models for the actors, and the Kinect device was used to capture the real time SCF dataset. As in this time, six basic emotional facial expressions were captured in 230 frames

from 30 actors. To reduce the computing load, the face area was separated into six parts, “left eye,” “left cheek,” “left mouth corner,” “right eye,” “right cheek,” and “right mouth corner,” as shown in Fig. 5.3.

5.1.4 Recognition Network

In this case, the network structure is recalled from Sec. 4.2 and Sec. 4.3.

Front-end Conversion

.

Table 5.2 The specified structure of convolutional autoencoder (CAE)

Layer	**Output Shape** Width, Height, Depth	**Parameters**	**Connected**
Input	96, 96, 1	9216	sMCAE output
Conv. 1	86, 86, 36	465948	Input
Ave. pooling	80, 80, 18	0	Conv. 1
Conv. 2	64, 64, 9	36864	Ave. pooling
Ave. pooling	28, 28, 9	0	Conv. 2
Flatten	1664	11741184	Ave. pooling
Dense	512	851968	Flatten
Dropout	384	0	Dense
Dense	100	38400	Dropout
Dense	6	600	Dense

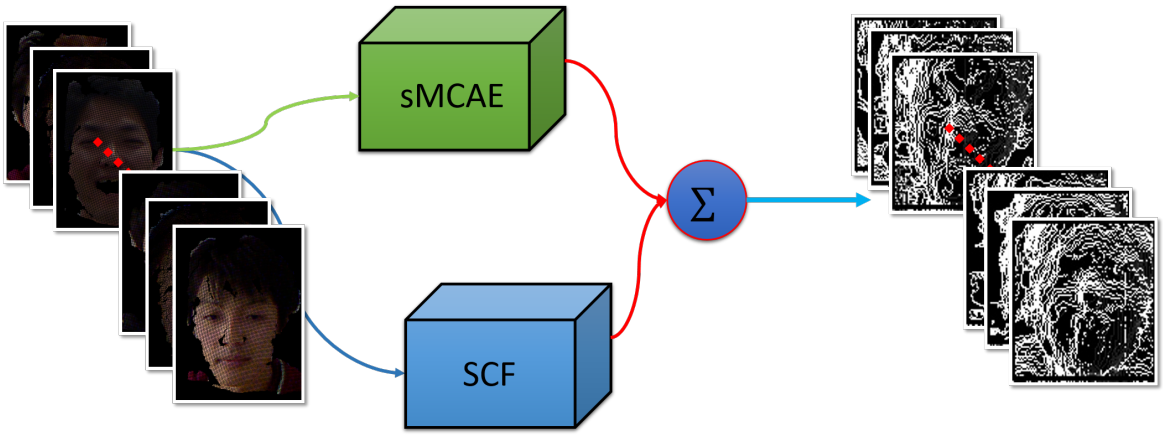

Fig. 5.4 The front-end convert DL structure

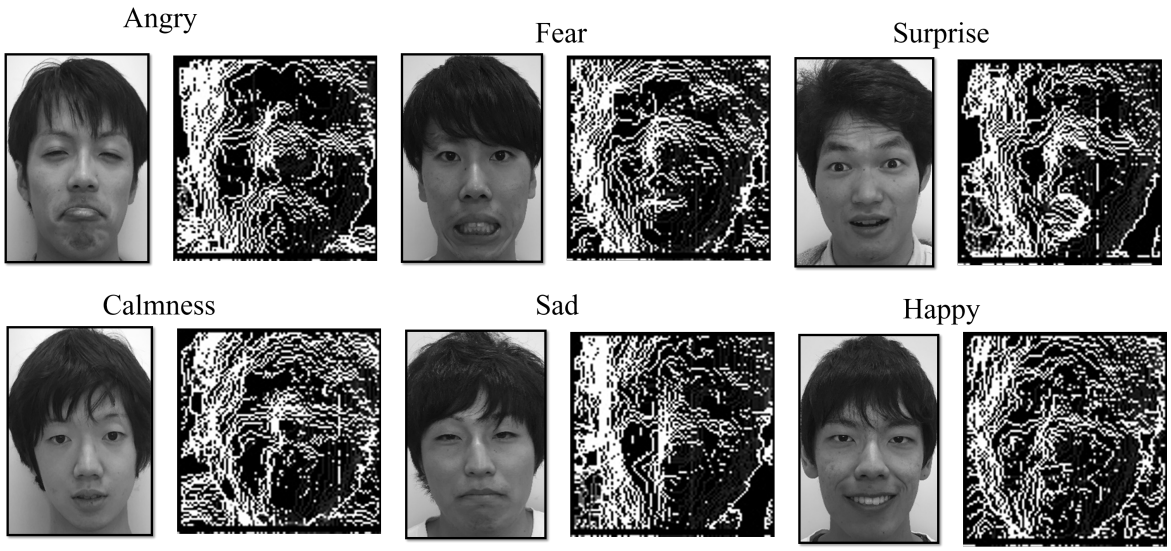

Fig. 5.5 The few results from the front-end network. Each emotion are shown with one RGB image (on the left) and one SCF map (on the right).

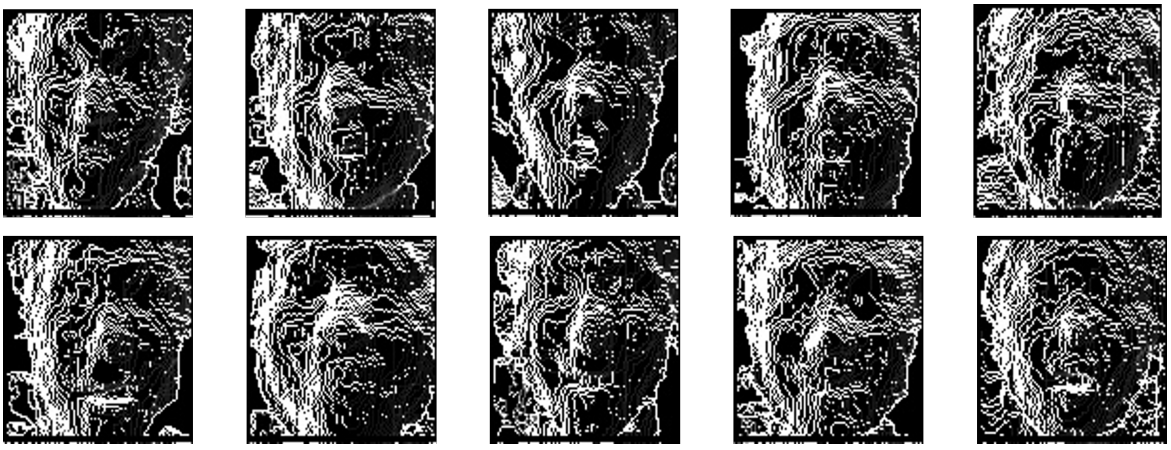

Fig. 5.6 Some results from the front-end network.

As shown in Fig. 5.4, the front-end part, that is the sMCAE provided from Sec. 4.2 and Sec. 4.3 is used, where the detail of the model is shown in Table 5.2. During the facial expression recognition task, the human face roughness is far small than the noise data, so that to gain such value, the SCF values and vibration values are summarized in the last of front-end layer. The converted results are shown in Fig. 5.5 and Fig. 5.6, where the SCF results from former represents the emotions, respectively.

Back-end Classification

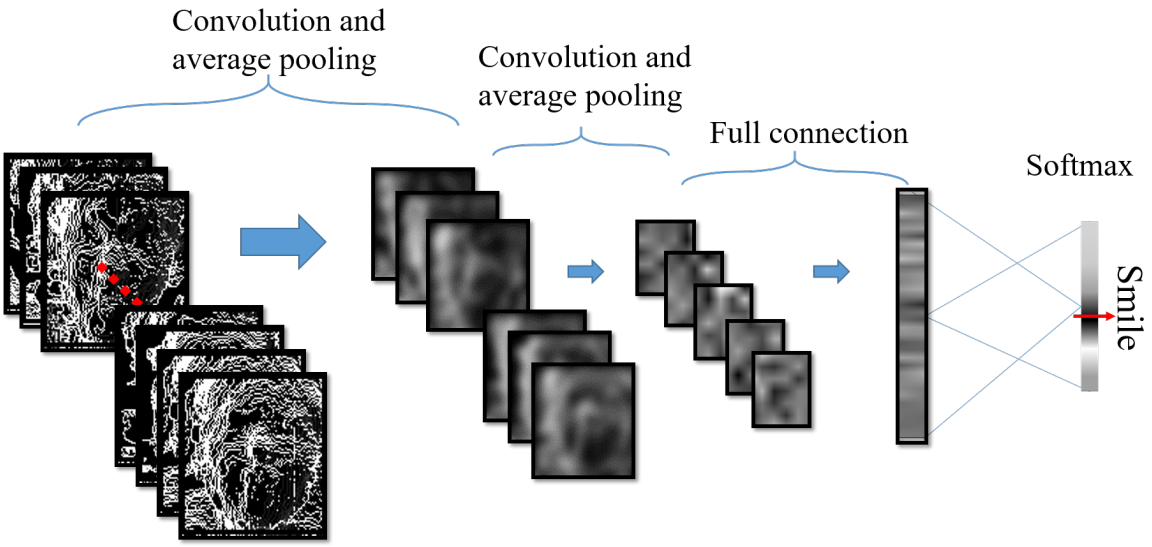

Fig. 5.7 The back-end concolutional neural network

As shown in Fig. 5.7, the back-end part, i.e., a typical CNN is used to classify the emotion by combining the output of front-end network. In this CNN, there are mainly three different type layers: the first type is a convolutional layer, which is located in the leftmost and third layers; the second type is an average pooling layer, which cames right after the convolutional layer; and the last type is a dense layer which is located in the rightmost to integrate the feature maps and predict the result.

5.1.5 Result

The left confusion map in Fig. 5.8 shows the recognition accuracy when there is only one of facial parts, whereas the right-hand side confusion matrix shows the result with the full facial parts. Fig. 5.9 shows the training and testing loss. The accuracy on the training and testing data is shown in Fig. 5.10. It is found from this figure that at around 100 epochs, the recognition accuracy is achieved nearly 97 percent.

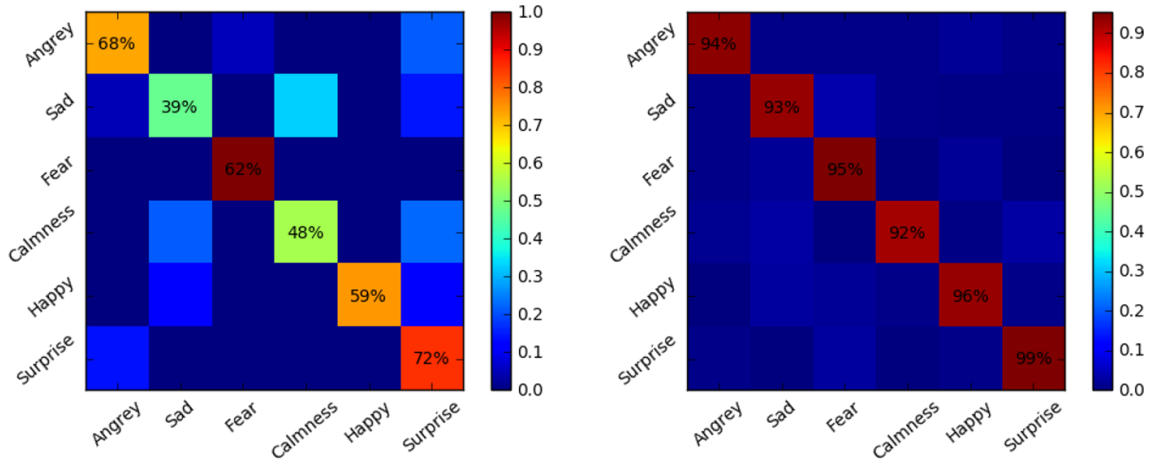

Fig. 5.8 A confusion map sample of left eye (left), and the confusion map of final prediction relied on the full facial parts (right)

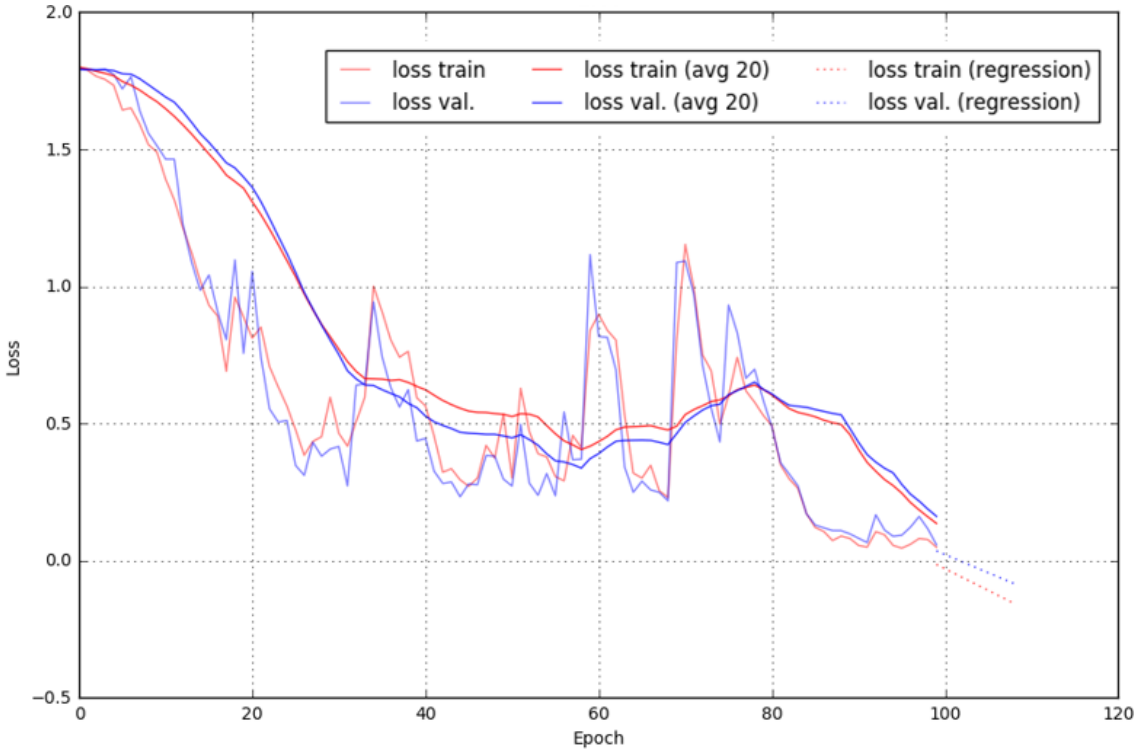

Fig. 5.9 The training and testing loss

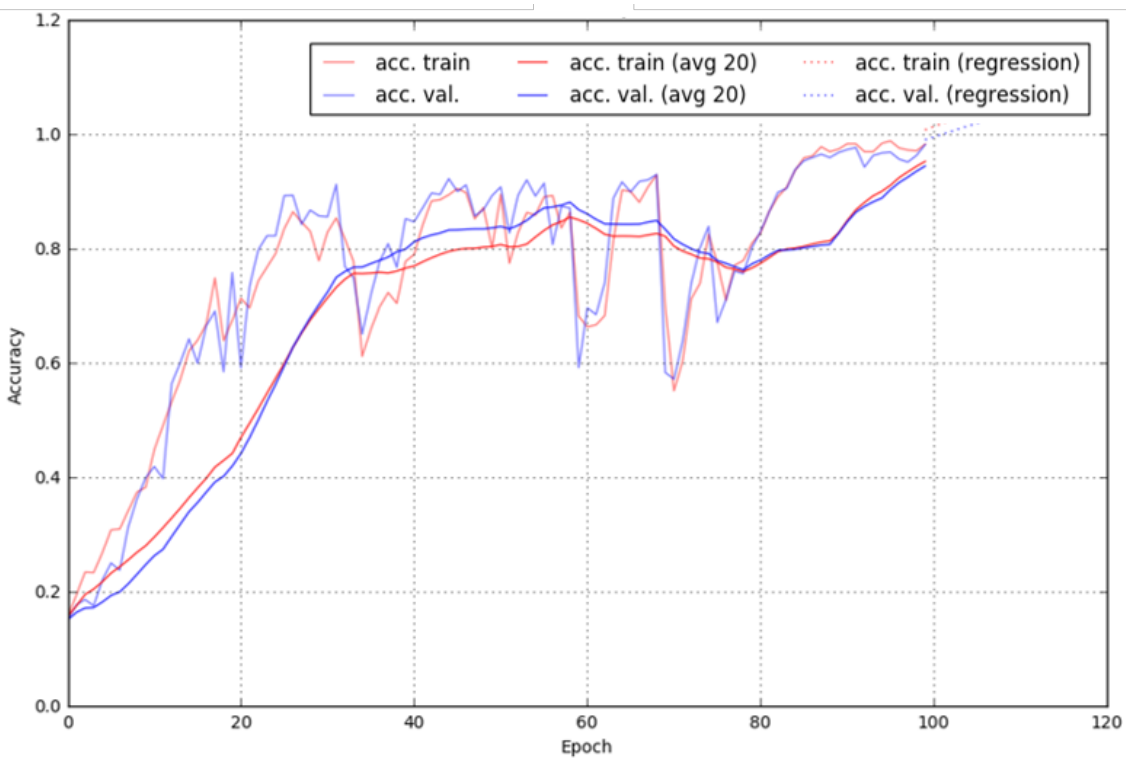

Fig. 5.10 The training and testing accuracy

5.2 Object Recognition

5.2.1 Introduction

Object recognition is one of the main subjects in computer vision, and also it is important for making robots useful in complex environments. As one of the most activated branch fields, most of the 3D object recognition [69-72] are limited in the conventional research to instance recognition (model-based key-point matching to nearest neighbors [73-78, 82]). Those methods also use hand-designed features such as SIFT, or HOG. In this section, we apply the previous DL model to object recognition. As shown in Fig. 5.11, the RGB-D object dataset [79, 81] is used to train our models, and some captured original data are used to test the model. And the compared result shows in Table 5.3.

5.2.2 Model Details

As shown in Fig. 5.13, Socher and Richard [84] proposed a 3D object recognition which is also based on the DL structure. Their model uses both raw RGB and depth images. First, it separately extracts features from RGB and depth. Each modality is first given to

Fig. 5.11 Some samples from RGB-Depth trainning dataset

Table 5.3 Comparison of our SCFnet to multiple related approaches.

Classifier	**Extra features for 3D and RGB**	**3D**	**RGB**	**Both**
Linear SVM	Spin Images, efficient match kernel (EMK), random Fourier sets, width, depth, height; SIFT, EMK, texton histogram, color histogram	53.1 ±1.7	74.3 ±3.3	81.9 ±2.9
Random Forest	same as above	64.7 ±2.3	83.9 ±3.5	83.9 ±3.5
CNN-RNN	Depth convolution filters; RGB convolution filters	78.9 ±3.8	82.4 ±3.1	86.8 ±3.3
SCFnet	sMCAE firing map, surface common features; RGB convolution filters	91.7 ±1.2	79.5 ±1.7	89.8 ±8.2

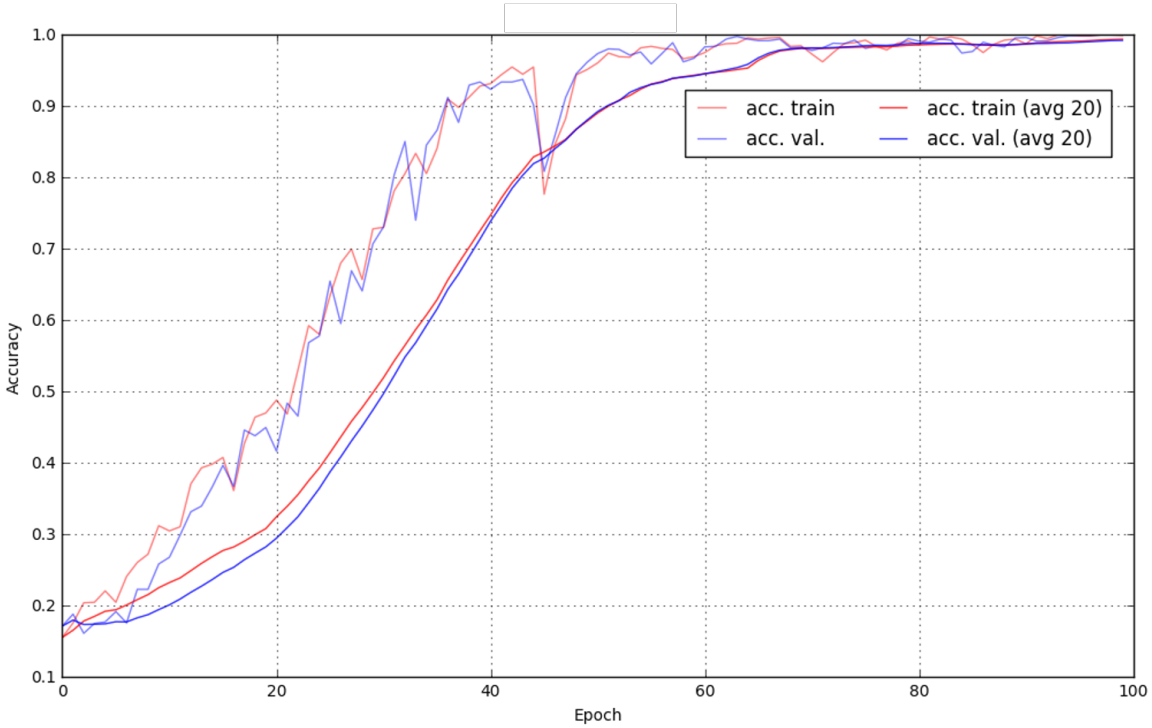

Fig. 5.12 Training result of SCFnet with RGB-D object dataset

a single convolutional layer which provides a useful translational invariance of low level features such as edges and allows the parts of an object to be deformable to some extent. The pooled filter responses are then given to a recurrent neural network (RNN), which can learn compositional features and part interactions. The RNNs hierarchi cally project inputs into a lower dimensional space through multiple layers with tied weights and nonlinearities [79, 80, 83]. In this experiment, we modified a new structure according to this model, but used our SCF based network (SCFnet) instead of their depth-images based CNN, where the structure is shown in Fig. 5.14.

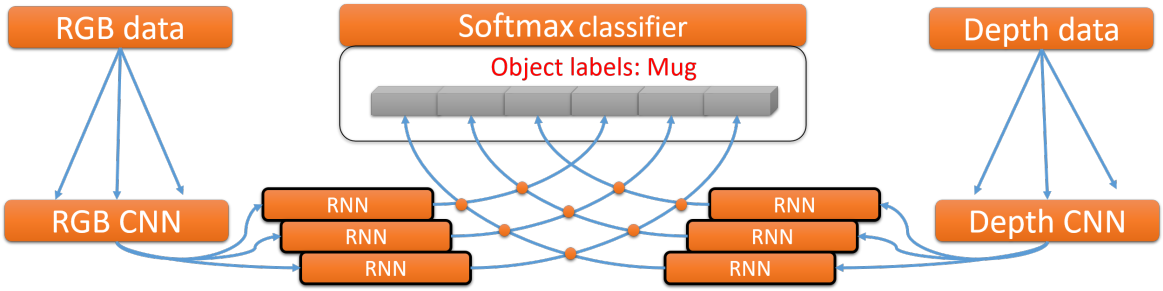

Fig. 5.13 The referred model

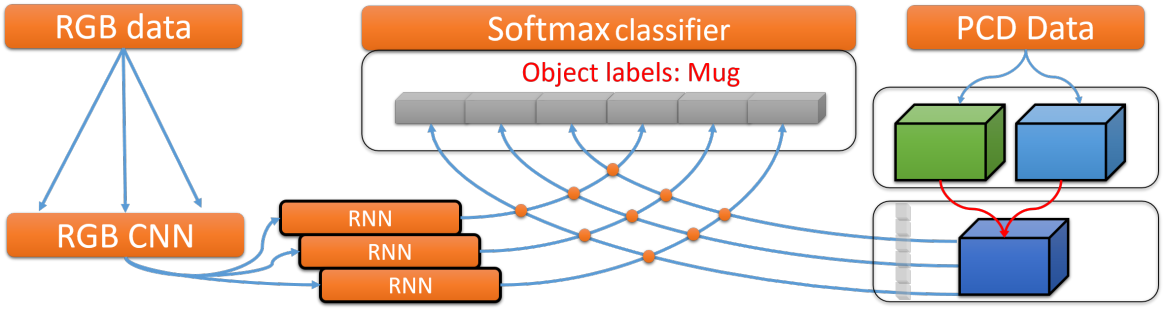

Fig. 5.14 Our modified model

5.2.3 Unsupervised Pre-training Filters

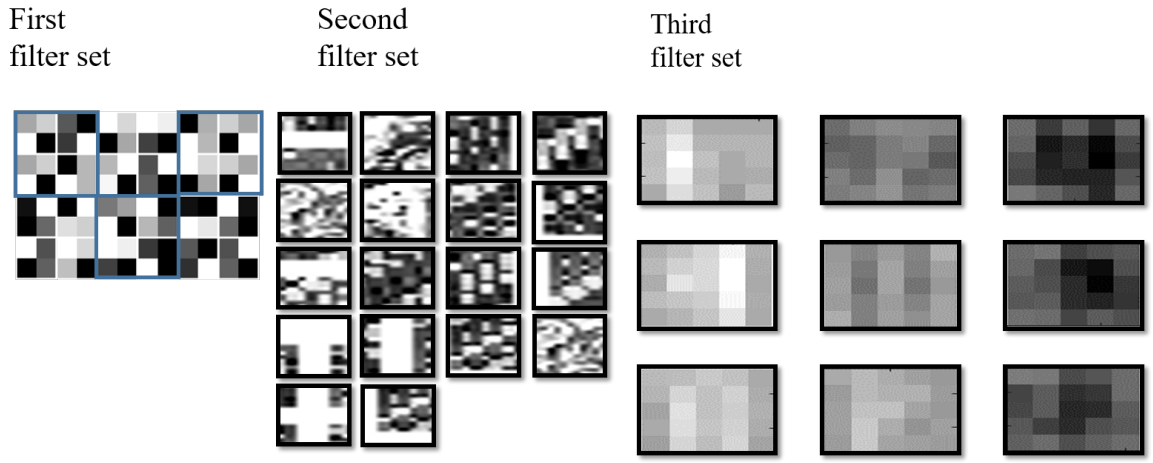

Fig. 5.15 The trained out filters of sMCAE.

Fig. 5.15 shows the trained out filters of the unsupervised sMCAE. To be more specific, the visualization of the back-end CNN filters are shown in Fig. 5.16 and Fig. 5.17.

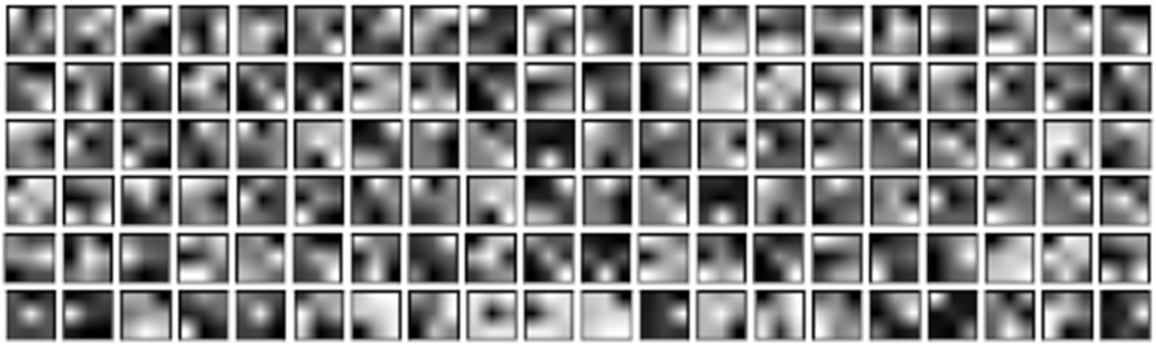

Fig. 5.16 The pre-trained filters of back-end CNN: i.e., the first convolution layer filters.

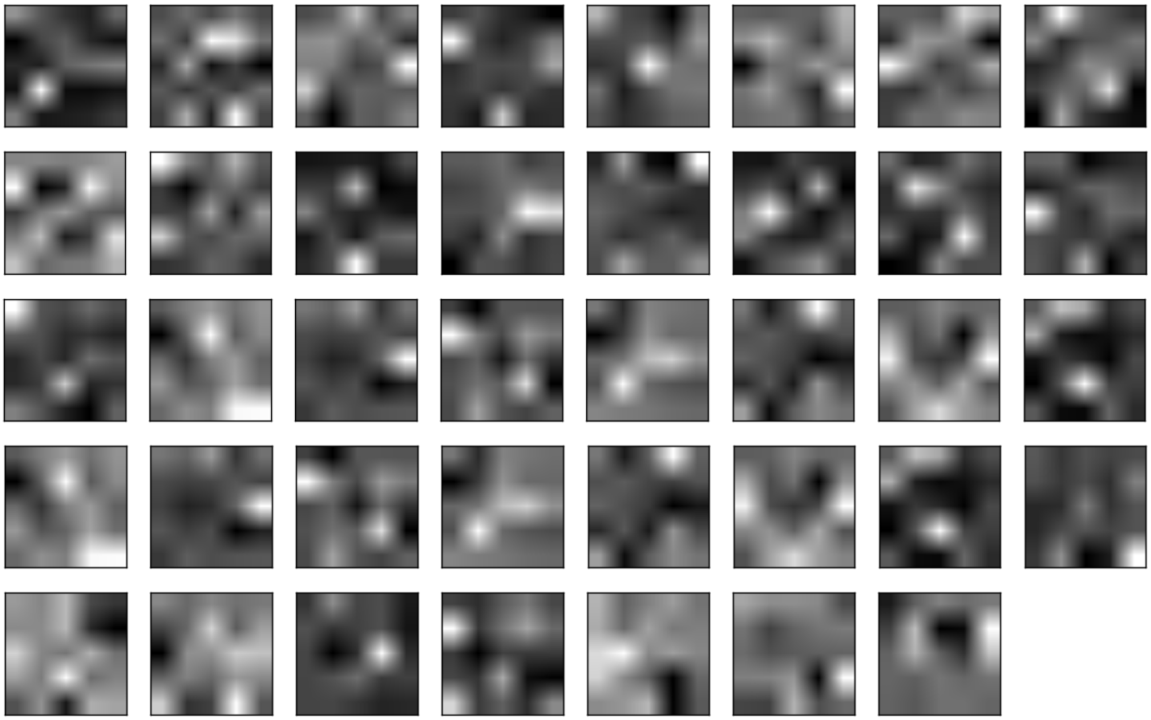

Fig. 5.17 The pre-trained filters of back-end CNN: i.e., the second convolution layer filters.

5.2.4 Object Detection

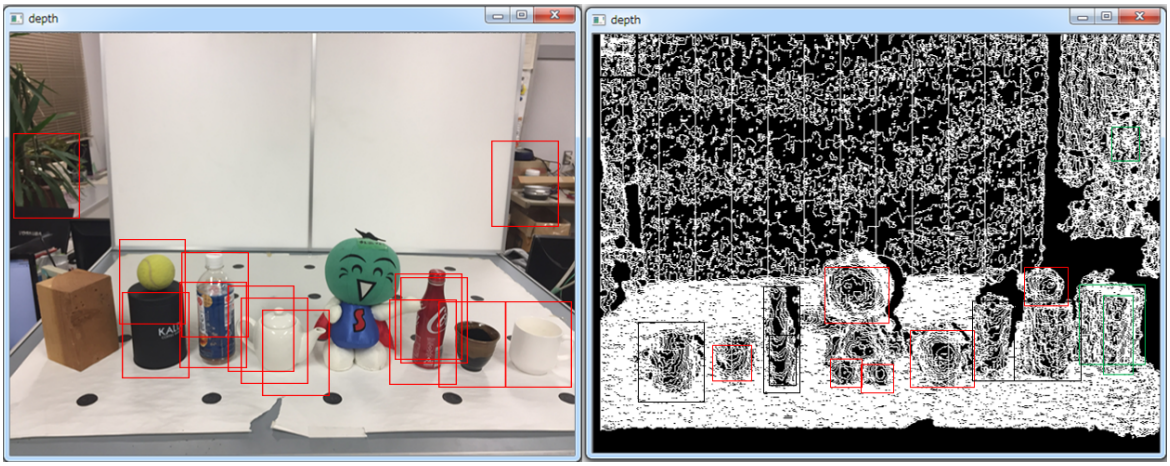

Fig. 5.18 The object detection task: where the woodblock and toy are not included in the training database.

To measure our modified SCFnet, some untrained objects are used to test the result that once the model faces unknown objects. As shown in Fig. 5.18, the system still can recognize the object by their geometrical features. The red blocks represent the objects are

only detected by the SCFnet (more than 60 percent of probability), and the black blocks represent the object are detected by both RGB and SCFnet. However, there is a woodblock, which confuse the SCFnet, but not the RGB network.

5.3 Summary

In this chapter, our DL structure has been applied on two tasks, i.e., a facial expression recognition and an object recognition. This model was also compared with other research to demonstrate that, the result showed increasingly high accuracy in both task.

Chapter 6

Conclusion

This thesis is concluded by summarizing our contributions. Finally, we discuss future research directions.

6.1 Contributions

Firstly, it developed unsupervised Deep Learning(DL) models for discovering features from Point Cloud Data (PCD) by proposing in different steps. While the popularity of DL has grown rapidly in many fields, the application of DL in 3D object recognition was previously limited to using depth map and other images-based methods to pre-processing the data, which lost most of the information that a 3D object could have. In contrast we developed new models based on human touch sensing theory that used surface roughness and surface conditions to produce the surface common features. The earlier methods were based on using Hidden Markov Model to recognize objects by their shape information and surface curvature. By calculating the vectors between two presented points, some basic surface curvatures were defined. This method produced nearly 70 percent of accuracy. However, this method with a new concept of curvature was still mainly based on images with too many pre-processing steps. Lately, a new method was proposed to present the PCD in a new structure. The method included two main networks as a full front-end network. We learned there are two main cells in human touch sensing: one is texture detection cells named Meissner corpuscles, which respond to the "feeling" of roughness in a surface by the vibration sensor cell; the other is Merkel cells, i.e., edge and surface curvature detection neurons, which respond to the skin indentation. Corresponding to this theory, the core approach of this front-end network included two parts: an stacked memories convolutional autoencoder (sMCAE) was designed to simulate the vibration neurons, and converted a PCD to the surface roughness; the other network was used to sense the skin indentations which

react to the object edge and surface curvature. We achieved a successful classification, and some supervised definitions of some simple classes were used to pre-train the network, such as upward inclined, downward inclined, upward curved, downward curved, edge and flat. Finally, we used a properly designed convolutional neural network (CNN) as the back-end network to merge the roughness information and surface condition information, and this CNN is also used to classify objects. As a result, the network finally was able to achieve 90 percent of accuracy. An emotional facial expression recognition task and an object recognition task were designed to test our model, which was developed in earlier chapters. However, there weren't any available PCD-based DL models, we compared our model with a RGB-Depth based DL model on object recognition task. Thus, our model showed significant improvements in the task.

6.2 Future Work

In this paper, feature learning was used to perform the semi-supervised training in specific objects. Due to the wideness of nature, there are millions of objects which are different to each other even in a same category. To be able to recognize unknown objects, it is very difficult if the artificial network is trained in specific objects. As the future work, the training procedure must include from low level features, edge, curvatures, and colors, to high level features, i.e., the meaning of the structure. Specifically, the definitions of a cup in human brain includes not only the object's color, shape or texture, but the object's significance, which is defined as connection features. In future research, the environment based memories network which considers the meaning of the geometrical features from the different environments, will be added in the DL structure.

References

[1] J. E. Johnson and Martial Hebert, "Using spin images for efficient object recognition in cluttered 3D scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 433–449, 1999.

[2] D. G. Lowe, "Local feature view clustering for 3D object recognition," in *Proceedings of Computer Vision and Pattern Recognition*, 2001, pp. 682–699.

[3] M. Liu and X. Li, "Generic object recognition based on the fusion of 2D and 3D SIFT descriptors," in *Proceedings of 18th International Conference on Information Fusion*, 2015, pp. 1085–1092.

[4] Y. Lei, M. Bennamoun, and M. Hayat, "An efficient 3D face recognition approach using local geometrical signatures," *Pattern Recognition*, vol. 47, no.2, pp. 509–524, 2014.

[5] A. Barr, P. R. Cohen, and E. A. Feigenbaum, *The handbook of Artificial Intelligence*. California, William Kaufmann, 1989.

[6] R. C. O'Reilly and Y. Munakata, *Computational Explorations in Cognitive Neuroscience: Understanding the Mind by Simulating the Brain*. MA, MIT press, 2000.

[7] J. Allanson and S. H. Fairclough, "A research agenda for physiological computing," *Interacting with computers*, vol. 16, no. 5, pp. 857–878, 2004.

[8] A. Girouard, T. Desney, N, Lennart, and M. Regan, "Brain, body and bytes: psychophysiological user interaction," in *Proceedings of CHI'10 Extended Abstracts on Human Factors in Computing Systems*, pp. 4433–4436, 2010.

[9] A. Calimera, M. Enrico, and P. Massimo, "The human brain project and neuromorphic computing," *Functional Neurology*, vol. 28, no. 3, pp. 191–196, 2013.

[10] J. O. Kephart, and M. C. David, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[11] T. B. Moeslund, H. Adrian, and K. Volker, "A survey of advances in vision-based human motion capture and analysis," *Computer Vision and Image Understanding*, vol. 104, no. 2, pp. 90–126, 2006.

[12] A. Jaimes and S. Nicu, "Multimodel human-computer interaction: A survey," *Computer Vision and Image Understanding*, vol. 108, no. 1, pp. 116–134, 2007.

[13] A. Konar, *Artificial Intelligence and Soft Computing: Behavioral and Cognitive Modeling of the Human Brain*. CRC press, 1999.

[14] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, Springer, 2006.

[15] G. Thomas, "Machine learning in ecosystem informatics and sustainability," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, 2009, pp. 8–13.

[16] C. Zhang and C. W. Philip, "Parametrized sigmoid and ReLU hidden activation functions for DNN acoustic modeling," in *Proceedings of Interspeech.* 2015, pp. 3224–3228.

[17] R. Salakhutdinov and G. E. Hinton, "Deep Boltzmann Machines," in *Proceedings of International Conference on Artificial Intelligence and Statistics.* 2009, pp 3–11.

[18] Y. Bengio, *Learning Deep Architectures for AI*. North America, now, pp. 1–127, 2009.

[19] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," *ICML Unsupervised and Transfer Learning,* vol. 27, pp. 37–50, 2012

[20] R. Socher, "Semi–supervised recursive autoencoders for predicting sentiment distributions," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics*, 2011, pp. 151–161.

[21] N. Srivastava and R. Salakhutdinov, "Multimodal learning with deep boltzmann machines," in *Advances in Neural Information Processing Systems*, 2012, pp. 2222–2230.

[22] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proceedings of the 25th International Conference on Machine learning*, 2008, pp. 1096–1103.

[23] Q. V. Le, J. Ngiam, A. Coates, and Ng Y. Andrew, "On optimization methods for deep learning," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 265–272.

[24] Z. Wu, Y. G. Jiang, J. Wang, and X. Xue, "Exploring inter–feature and inter–class relationships with deep neural networks for video classification," in *Proceedings of the 22nd ACM International Conference on Multimedia*, 2014, pp. 167–176.

[25] J. Masci, U. Meier, D. Ciresan, and J. Schmidhuber, "Stacked convolutional auto–encoders for hierarchical feature extraction," in *Proceedings of International Conference on Artificial Neural Networks*, 2011, pp. 52–59.

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

[27] P. Vincent, H. Larochelle, I. Lajoie, and Y. Bengio, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, pp. 3371–3408, 2010.

[28] A. R. Abbas, K. Wolslegel, D. Seshasayee and Z. Modrusan, "Deconvolution of blood microarray data identifies cellular activation patterns in systemic lupus erythematosus," *PloS one*, vol. 4, no. 7, e6098, 2009.

[29] L. Deng, "A tutorial survey of architectures, algorithms, and applications for deep learning," *APSIPA Transactions on Signal and Information Processing*, vol. 3, e2, 2014.

[30] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[31] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[32] M. E. Yumer, P. Asente, R. Mech, and L. B. Kara, "Procedural Modeling Using Autoencoder Networks," in *Proceedings of the 28th Annual ACM Symposium on User Interface Software and Technology*, 2015, pp. 109–118.

[33] H. Kamyshanska and R. Memisevic, "The potential energy of an autoencoder," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 6, pp. 1261–1273, 2015.

[34] A. Mannini and A. M. Sabatini, "Machine learning methods for classifying human physical activity from on–body accelerometers," *Sensors*, vol. 10, no. 2, pp. 1154–1175, 2010.

[35] R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*. Berlin, Springer, 1983.

[36] R. B. Rusu and C. Steve, "3d is here: Point cloud library (pcl)," in *Proceedings of International Conference on Robotics and Automation*, 2011, pp. 1–4.

[37] A. Boulch and M. Renaud, "Fast and robust normal estimation for point clouds with sharp features," *Computer Graphics Forum*, vol. 31, no. 5, pp. 1766–1774, 2012.

[38] H. Hoppe, T. DeRose, T. Duchamp, and J. McDonald, "Surface reconstruction from unorganized points," *ACM SIGGRAPH Computer Graphics*, vol. 26, no. 2, 1992.

[39] J. Zhang, J. Cao, X. Liu, J. Wang, J. Liu, and X. Shi, "Point cloud normal estimation via lowrank subspace clustering," *Computers and Graphics*, vol. 37, no. 6, pp. 697–706, 2013.

[40] G. Guennebaud and M. Gross, "Algebraic point set surfaces," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, pp. 23, 2007.

[41] D. L. Page, Y. Sun, A. F. Koschan, J. Paik, and M. A. Abidi, "Normal vector voting: crease detection and curvature estimation on large, noisy meshes," *Graphical Models*, vol. 64, no. 3, pp.199–229, 2003.

[42] T. K. Dey, G. Li, and J. Sun, "Normal estimation for point clouds: A comparison study for a Voronoi based method," in *Proceedings of Eurographics/IEEE VGTC Symposium Point–Based Graphics*, 2005, pp. 39–46.

[43] Q. Merigot, O. Maks, and J. G. Leonidas, "Voronoi-based curvature and feature estimation from point clouds," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 6, pp. 743–756, 2011.

[44] X. P. Zhang, H. J. Li, and Z. L. Cheng, "Curvature estimation of 3D point cloud surfaces through the fitting of normal section curvatures," in *Proceedings of AsiaGraph*, 2008, pp. 23–26.

[45] S. Holzer, R. B. Rusu, M. Dixon, and S. Gedikli, "Adaptive neighborhood selection for real–time surface normal estimation from organized point cloud data using integral images," in *Proceedings of International Conference on Intelligent Robots and Systems*, 2012, pp. 2684–2689.

[46] N. J. Mitra and A. Nguyen, "Estimating surface normals in noisy point cloud data," in *Proceedings of the 19th Annual Symposium on Computational Geometry*, 2003, pp. 322–328.

[47] Y. Guo, M. Bennamoun, F. Sohel, and M. Lu, "3D object recognition in cluttered scenes with local surface features: a survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 11, pp. 2270–2287, 2014.

[48] D. Zheng, J. Xu, and C. Ren-xi, "Generation method of normal vector from disordered point cloud,"in *Proceedings of IEEE Joint Urban Remote Sensing Event*, 2009, pp. 1–5.

[49] X. D. Yang and Y. L. Tian, "Super normal vector for activity recognition using depth sequences," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 804–811.

[50] D. OuYang and H. Y. Feng, "On the normal vector estimation for point cloud data from smooth surfaces," *Computer–Aided Design*, vol. 37, no. 10, pp. 1071–1079, 2005.

[51] I. M. Park, E. W. Archer, and N. Priebe, "Spectral methods for neural characterization using generalized quadratic models," in *Proceedings of Advances in Neural Information Processing Systems*, 2013, pp. 2454–2462.

[52] P. Dayan and L. F. Abbott, "Theoretical neuroscience: computational and mathematical modeling of neural systems," *Journal of Cognitive Neuroscience*, vol. 15, no. 1, pp. 154–155, 2003.

[53] Z. Yan, W. Zhang, Y. He, D. Gorczyca, Y. Xiang, and L. E. Cheng, "Drosophila NOMPC is a mechanotransduction channel subunit for gentle–touch sensation," *Nature*, vol. 493, no. 7431, pp. 221–225, 2013.

[54] T. J. Prescott, E. D. Mathew, and M. W. Alan, "Active touch sensing," *Philosophical Transactions of the Royal Society B*, vol. 366, no. 1581, pp.2989–2995, 2011.

[55] Y. Roudaut, A. Lonigro, B. Coste, J. Hao, and P, Delmas, "Touch sense: functional organization and molecular determinants of mechanosensitive receptors," *Channels*, vol.6, no. 4, pp. 234–245, 2012.

[56] A. Gallace and C. Spence, "Touch and the Body," *Psyche*, vol. 16, no. 1, pp. 30–67, 2010.

[57] G. Robles-De-La-Torre, "The importance of the sense of touch in virtual and real environments," *IEEE Multimedia*, vol. 13, no. 3, pp. 24–30, 2006.

[58] M. Pantic and P. Ioannis, "Dynamics of facial expression: recognition of facial actions and their temporal segments from face profile image sequences," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 31, no. 2, pp. 433–449, 2006.

[59] Z. Zeng, M. Pantic, and G. I. Roisman, "A survey of affect recognition methods: Audio, visual, and spontaneous expressions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol.31, no. 1, pp. 39–58, 2009.

[60] J. J. J. Lien, T. Kanade, J. F. Cohn, and C. C. Li, "Detection, tracking, and classification of action units in facial expression," *Robotics and Autonomous Systems*, vol.3, pp. 131–146, 2000.

[61] Y. Tong, J. Chen, and Q. Ji, "A unified probabilistic framework for spontaneous facial action modeling and understanding," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 3, pp. 258–273, 2010.

[62] Y. Tian, T. Kanade, and J. F. Cohn, "Recognizing lower face action units for facial expression analysis," in *Proceedings of 4th International Conference on Automatic Face and Gesture Recognition*, 2000, pp. 484–490.

[63] Y. Tian, T. Kanade, and J. F. Cohn, "Recognizing action units for facial expression analysis," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 23, no. 2, pp. 97–115, 2001.

[64] M. S. Bartlett, G. Littlewort, and M. Frank, "Recognizing facial expression: machine learning and application to spontaneous behavior," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, pp. 568–573.

[65] T. Kanade, J. F. Cohn, and Y. Tian, "Comprehensive database for facial expression analysis." in *Proceedings of 4th International Conference on Automatic Face and Gesture Recognition*, 2000, pp. 46–53.

[66] R. A. James, "The circumplex model of affect," *Journal of Personality and Social Psychology*, vol. 39, no. 6, pp. 1161–1178, 1980.

[67] J. Posner, R. A. James, and S. P. Bradley, "The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology," *Development and psychopathology*, vol. 17, no. 3, pp. 715–734, 2005.

[68] P. Ekman, "Facial expression and emotion," *American Psychologist*, vol. 48, no. 4, p. 384, 1993.

[69] R. Adolphs, "Neural systems for recognizing emotion," *Current Opinion in Neurobiology*, vol. 12, no. 2, pp. 169–177, 2002.

[70] M. T. Posamentier and A. Herve, "Processing faces and facial expressions," *Neuropsychology Review*, vol. 13, no. 3, pp. 113–143, 2003.

[71] R. J. James and W. Davidson, *Handbook of Affective Sciences*. Oxford University Press, 2002.

[72] P. Ekman and F. V. Wallace, "Measuring facial movement." *Environmental Psychology and Nonverbal Behavior*, vol. 1, no. 1, pp. 56–75, 1976.

[73] H. Chen and B. Bir, "3D free–form object recognition in range images using local surface patches," *Pattern Recognition Letters*, vol. 28, no. 10, pp. 1252–1262, 2007.

[74] Z. Teng and X. Jing, "Surface–based general 3D object detection and pose estimation," in *Proceedings of IEEE International Conference on Robotics and Automation*, 2014, pp. 5473–5479.

[75] K. Tanaka, "Inferotemporal cortex and object vision," *Annual Review of Neuroscience*, vol. 19, no. 1, pp. 109–139, 1996.

[76] F. Rothganger, S. Lazebnik, and C. Schmid, "3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints," *International Journal of Computer Vision*, vol. 66, no. 3, pp. 231-259, 2006.

[77] A. E. Johnson and H. Martial, "Using spin images for efficient object recognition in cluttered 3D scenes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 5, pp. 433–449, 1999.

[78] A. K. Jain and D. Chitra, "3d object recognition," *Statistics and Computing*, vol. 10, no. 2, pp. 167–182, 2000.

[79] K. Lai and D. Fox, "Object recognition in 3D point clouds using web data and domain adaptation," *The International Journal of Robotics Research*, vol. 29, no. 8, pp. 1019–1037, 2010.

[80] K. Lai, L. Bo, X. Ren, and D. Fox, "Detection-based object labeling in 3D scenes," in *Proceedings of International Conference on Robotics and Automation*, 2012, pp. 1330–1337.

[81] G. Pang and U. Neumann, "Training–based object recognition in cluttered 3d point clouds," in *Proceedings of International Conference on 3D Vision*, 2013, pp. 87–94.

[82] J. Huang and S. You, "Detecting objects in scene point cloud: A combinational approach," in *Proceedings of International Conference on 3D Vision*, 2013, pp. 175–182.

[83] K. Lai, L. Bo, X. Ren, and D. Fox. "A large-scale hierarchical multi-view rgb-d object dataset," in *Proceedings of International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1817–1824.

[84] R. Socher, B. Huval, B. P. Bath, C. D. Manning, and A. Y. Ng, "Convolutional-recursive deep learning for 3d object classification," *Neural Information Processing Systems*, vol. 3, no. 7, pp. 665-673, 2012.

Appendix A

A Raspberry Pi 3 Powered High Performance Computing for SCFnet

The world's fastest computer is currently China's Sunway TaihuLight with 10,649,600 CPU cores. The 32 cores created by 8 Raspberry Pi 3 units might seem less impressive, but it's an enough good computing environment for a DL with nearly 30 layers (200k parameters), where the requirments are shown in Table A.1. In our case, we used 8 Raspberry Pi 3 to built a 32 core beowolf cluster system. The advantage of cluster is that, the one single Pi 3 is far lower than a desktop system with Core I7 cpu, but once applying the parallel computing technique, a 32 core cluster is more capable than a single Core I7.

In our DL training processing, the master node of this cluster works as the broadcast point, which is also gathering the information from Kinect, and checks the standby worker nodes, using the message passing interface to seperate the calculating gradient of neurons to each worker, as shown in Fig. A.1.

Table A.1 The required items for a Beowolf cluster system

Device	**Number**	**Other**
Raspberry Pi 3	8	
Ethernet Hub/Router	1	at least 9 ports
TF SD-cards	8	at least 16 GB
Kinect v1	1	v2 is not supported by Linux OS
Power supply	1	80 W

By creating this computing cluster, we were able to reduce the 70 percent of training time in this 30 layer DL structure (i.e., SCFnet), which was nearly 17 hours. And Fig. A.2 shows the assembled final cluster.

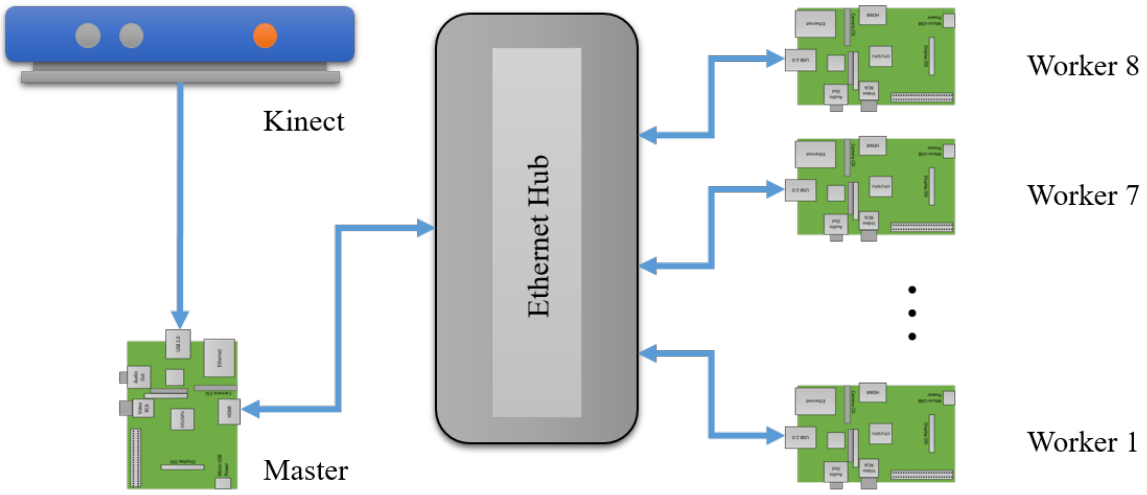

Fig. A.1 The Raspberry Pi based high performance computing cluster

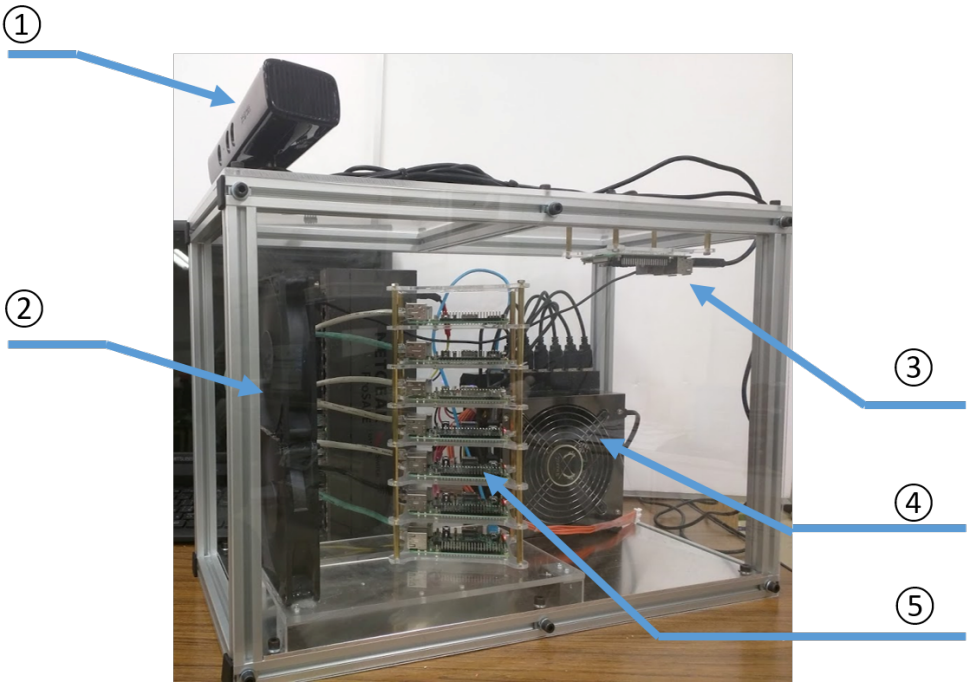

Fig. A.2 Assembled HPC, and some of the components: number 1 is the Kinect device; number 2 is two fans; number 3 is the master node; number 4 is the 80W power supply; and number 5 is the worker cluster.

Appendix B

Object Data

This appendix shows the data structure which is related to HMM-based recognition in Chapter 3. The data includes 3 main types of objects, i.e., cups, teapots, and bottles. Each of the objects have 10 different CAD source models. Each of the models is captured by applying the omnidirectional projection to produce the edge information and the information on the surface condition. The illustration of the data structure is shown in Fig. B.1, where the blue blocks represent the labels for different objects.

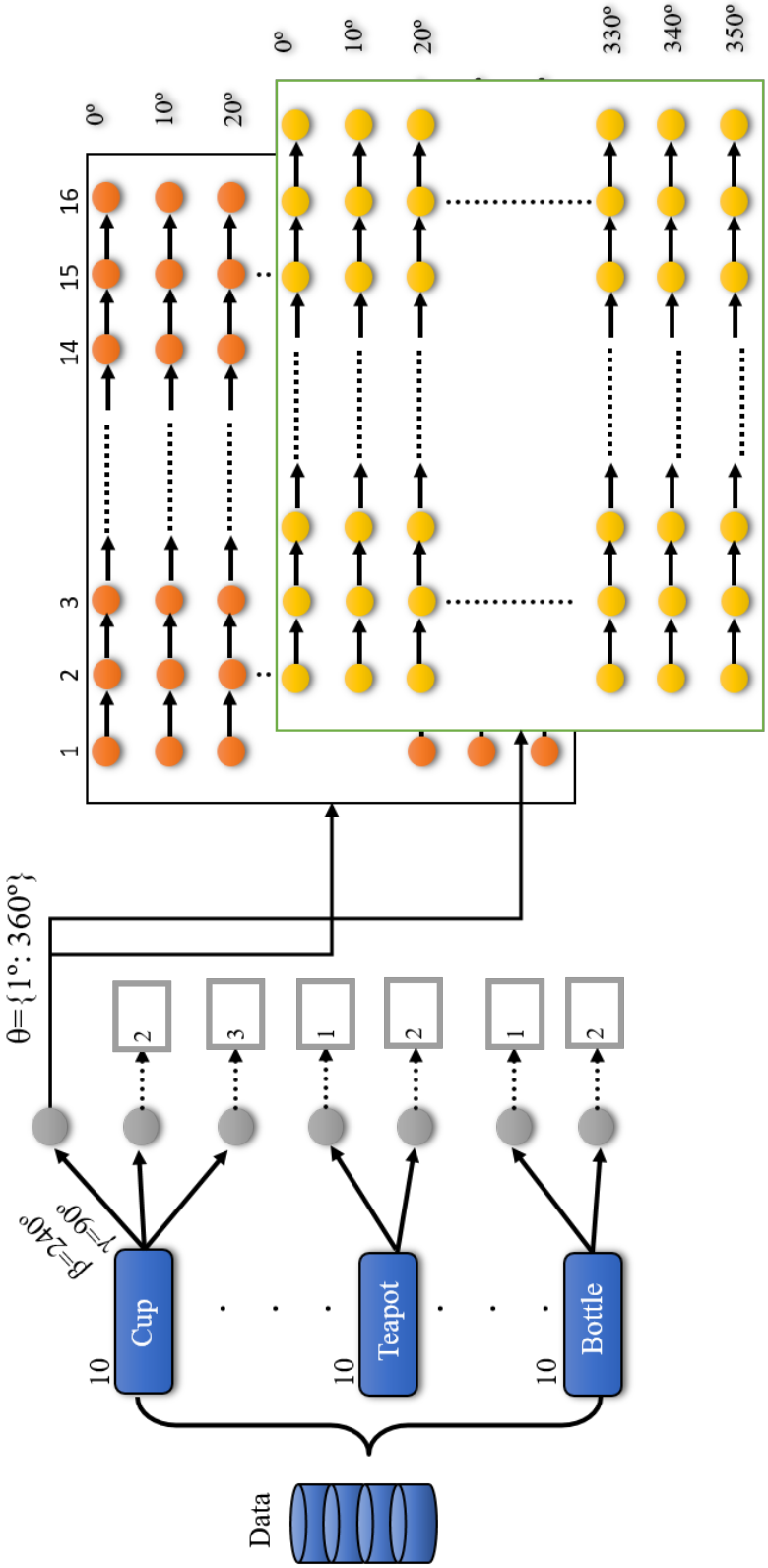

Fig. B.1 Illustrated detail of our model