RESEARCH-ARTICLE

A Real-Time Hand Posture Recognition System Using Deep Neural Networks

AO TANG, University of Science and Technology of China, Hefei, Anhui, China

KE LU, University of Chinese Academy of Sciences, Beijing, China

YUFEI WANG, University of Science and Technology of China, Hefei, Anhui, China

JIE HUANG, University of Science and Technology of China, Hefei, Anhui, China

HOUQIANG LI, University of Science and Technology of China, Hefei, Anhui, China

Published: 31 March 2015
Accepted: 01 January 2014
Revised: 01 November 2013
Received: 01 July 2013

A Real-Time Hand Posture Recognition System Using Deep Neural Networks

AO TANG, University of Science and Technology of China
KE LU, University of the Chinese Academy of Sciences
YUFEI WANG, JIE HUANG, and HOUQIANG LI, University of Science and Technology of China

Hand posture recognition (HPR) is quite a challenging task, due to both the difficulty in detecting and tracking hands with normal cameras and the limitations of traditional manually selected features. In this article, we propose a two-stage HPR system for Sign Language Recognition using a Kinect sensor. In the first stage, we propose an effective algorithm to implement hand detection and tracking. The algorithm incorporates both color and depth information, without specific requirements on uniform-colored or stable background. It can handle the situations in which hands are very close to other parts of the body or hands are not the nearest objects to the camera and allows for occlusion of hands caused by faces or other hands. In the second stage, we apply deep neural networks (DNNs) to automatically learn features from hand posture images that are insensitive to movement, scaling, and rotation. Experiments verify that the proposed system works quickly and accurately and achieves a recognition accuracy as high as 98.12%.

Categories and Subject Descriptors: I.4.6 [**Image Processing and Computer Vision**]: Segmentation—*Region growing, partitioning*; I.4.8 [**Image Processing and Computer Vision**]: Scene Analysis—*Range data, tracking*; I.5.4 [**Pattern Recognition**]: Applications—*Computer vision*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Kinect, hand tracking, posture recognition, deep neural networks

ACM Reference Format:
Ao Tang, Ke Lu, Yufei Wang, Jie Huang, and Houqiang Li. 2015. A real-time hand posture recognition system using deep neural networks. ACM Trans. Intell. Syst. Technol. 6, 2, Article 21 (March 2015), 23 pages.
DOI: http://dx.doi.org/10.1145/2735952

1. INTRODUCTION

Hand posture recognition (HPR) has been recognized as a valuable technology for several application fields, especially for sign language recognition (SLR). Sign languages consist of lots of complex hand movements, and every tiny posture can have a variety of possible meanings. That has made the precise recognition of tiny hand movements a major research concern for years, and the difficulties are twofold. One obstacle lies in the detecting and tracking of hands. Digital gloves or marker-based methods provide feasible solutions [Gao et al. 2004], but wearing extra equipment on the hands is

This work was supported by the Natural Science Foundation of China (NSFC) under contract No. 61325009, No. 61390514, and No. 61272316.
Authors' addresses: A. Tang, Y. Wang, J. Huang, and H. Li (corresponding author), Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei, China, 230027; emails: {tangao, feiyuw, jiehag}@mail.ustc.edu.cn, lihq@ustc.edu.cn; K. Lu, University of the Chinese Academy of Sciences, Beijing, China, 100098; email: luk@ucas.ac.cn.

DOI: http://dx.doi.org/10.1145/2735952

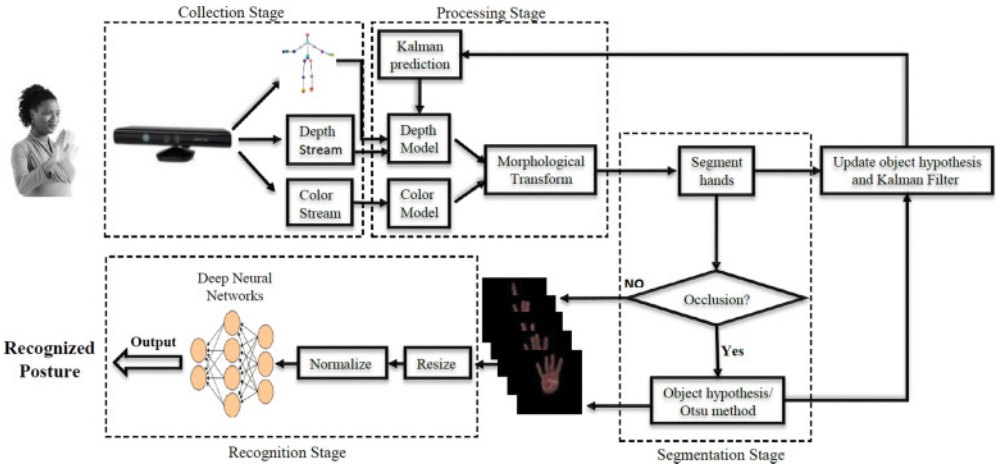

Fig. 1. Online framework of proposed posture recognition system.

inconvenient to people. Another noninvasive alternative, the vision-based methods, is limited by the poor performance of conventional cameras. Another difficulty arises from the contradiction that simple hand features usually lead to ambiguities, whereas sophisticated features cost more processing time, due to the complexity of hand postures. It's nontrivial to select what features to use for hand postures.

The development of HPR has been much promoted by two recent advances. The first is the release of Kinect by Microsoft. The device features an RGB camera, a depth sensor, and a multiarray microphone. It is capable of tracking users' body movements. With a Kinect, we can get both color and depth information and can enjoy a better experience in Human–Computer Interaction (HCI). Kinect has been successfully used in face recognition [Li et al. 2013] and action recognition [Xia et al. 2012]. However, though Kinect works well on tracking a large object such as a human body, it proves very challenging to directly detect and segment objects as small as hands from the video stream, due to the low resolution of its depth map. The second advance is the prosperity of the deep neural network (DNN), also called deep learning or feature learning. It provides a feasible solution for feature extraction and selection. By simulating the functionalities of the human brain, the DNN automatically analyzes and extracts features from raw data. It has been successfully applied to speech recognition [Seide et al. 2011] and image retrieval [Krizhevsky and Hinton 2011].

In this article, we propose a hand posture recognition system, which uses Kinect to detect and track human hands, followed by recognizing the hands' postures using deep learning. A framework is shown in Figure 1. In our system, we contribute to overcoming the following difficulties for hand posture recognition:

—Detecting and segmenting hands from complex backgrounds with noise and different illuminations
—Tracking hands in real time and automatically recovering from tracking failures
—Extracting robust features and precisely recognizing hand postures

More specifically, we enhance the performance of hand posture recognition from two sides. On one side, we present a new hand detection algorithm using Kinect. The algorithm combines both color and depth information and precisely finds the location of human hands. Then a robust hand tracking algorithm is presented, which can track

hands even with the occurrence of occlusions either between hands or between hands and the head. On the other side, after segmenting hands, we feed segmented hand images to DNNs directly instead of extracting and selecting features manually. Both deep belief networks and convolution neural networks have been used for hand posture recognition, and both of them achieve a result with very high accuracy. The proposed system works quickly in all detection, tracking, and recognition stages, which ensures that the system works in real time.

The remainder of this article is organized as follows. Related work is discussed in Section 2. In Section 3, we present our algorithms on hand detection and tracking using Kinect. Section 4 describes the deep neural networks for hand posture recognition. Experiments in Section 5 certify the effectiveness of our proposed system. Section 6 concludes the article.

2. RELATED WORK

2.1. Prior Work on Hand Detection and Segmentation

Many approaches have been tried to tackle the robust hand detection and segmentation problem, most of which fall into three categories: the color-based method, the depth-based method, and the method combining both color and depth information.

Color-based methods investigate the difference between the colors of the hand and the background. A straightforward skin color model is established in Fagiani et al. [2013], where skin regions are simply found by thresholding in YC_bC_r color space. Argyros and Lourakis [2004] model the probability distribution of skin color with a histogram constituted by statistics. Van den Bergh and Van Gool [2011] further incorporate a pretrained Gaussian mixture model to the histogram-based model. However, such methods often encounter the overlap between the hand and head or tend to confuse the hand with objects of a similar color. It is obvious that if the color of the background is very close to that of hand skin, those methods are very likely to fail. Seeking higher robustness, some researcher (e.g., [Caputo et al. 2012]) utilize gloves with a specific color that is easier to detect. A few additional constraints are also taken into account [Radha and Krishnaveni 2009]. Zabulis et al. [2009] try to use background subtraction to reduce the detection range. Chen et al. [2003] rely on edge detectors as well as motion detectors. However, the success of color-based methods still hinges a lot on the property of background color and stability.

Depth-based methods distinguish the difference between the depth of the hand and the background. A time-of-flight (ToF) camera is a frequently used range camera to acquire depth information [Van den Bergh and Van Gool 2011]. Microsoft Kinect provides an inexpensive way to obtain depth data. Despite the low resolution, prior work [Ren et al. 2011; Kurakin et al. 2012; Li 2012; Caputo et al. 2012] achieved good segmentation results using Kinect. However, these segmentation techniques assume that the hands are the closest objects to the camera while somewhat distant from the body, which is typically not the case in sign language.

Inspired by the pros and cons of both previously mentioned methods, we propose to combine both color and depth information, which leads to superior performance. Van den Bergh and Van Gool [2011] use a ToF camera and an RGB camera to allow hands overlapping with the face; however, calibration is necessary for a mapping from depth data to an RGB image. Caputo et al. [2012] use multiple Kinects to track the hand and an HD color sensor to segment the hand for stable gesture recognition. That, however, requires the user to wear colored gloves. In Ren et al. [2011], depth information is first used to find the frontmost object from the sensor, which is considered to be the hand and arm part, however, the user should wear a black belt on the wrist to help segment the hands from the arm.

2.2. Prior Work on Hand Posture Recognition

The recognition of objects in video has long been a challenging task, with various approaches proposed. A typical real-time object recognition and tracking system is discussed in Foresti [1999]. In order to exploit the wealth of unlabeled training data, a semisupervised method is introduced in Wang et al. [2009]. Wang et al. [2012] suggest that several crucial bottlenecks in video annotation and recognition can be simultaneously dealt with by learning with multiple graphs. More specifically, the recognition of hand postures has been well studied as a particular scene. Existing methods fall into two categories: the feature-based method and the appearance-based method.

Many feature-based methods focus on the number and the position of fingers. Cao and Li [2010] propose an HPR algorithm based on topological features. Topological features are easy to be calculated and stable even for an undefined person and hand rotation. In their experiments, they achieve a high recognition accuracy of 95.4% tested on nine postures. Aksaç et al. [2011] develop a virtual mouse system that can recognize hand postures. Convexity analysis of contours is employed to find fingers. The system can only recognize a few simple hand postures. Tusor and Varkonyi-Koczy [2010] recognize hand postures by extracting feature points that include curvature extrema, peaks (e.g., fingertips), and valleys (e.g., roots of the fingers). The selected feature points are matched in a stereo image pair using a fuzzy neural work. Although fuzzy neural networks are useful for recognition, it takes too much time to train the networks to obtain good results. The disadvantage of feature-based methods is that the number of postures recognized is very limited.

Besides feature-based methods, several appearance-based methods are used for HPR. Suryanarayan et al. [2010] present a method for detecting hand poses using 3D volumetric shape descriptors. The strength of their method lies in recognizing scale- and rotation-invariant hand poses. The accuracy is between 80% and 95% tested on six hand poses. Malassiotis and Strintzis [2008] demonstrate a system that relies on range data for recognition of static hand postures; the classification of hand postures is achieved by representing the range images by a discriminative feature vector that incorporates 3D shape information. The accuracy ranges from 70% to 90% on 20 hand postures.

2.3. Gesture Dataset Captured by Kinect

There exists several public 3D gesture recognition or sign language datasets captured by Kinect. The 2013 Multimodal Challenge workshop by ChaLearn[1] provides a dataset of 20 gestures of Italian signs. The dataset contains not only RGB-Depth-Skeleton data but also audio data. The MSRGesture3D dataset [Kurakin et al. 2012] contains 12 dynamic American Sign Language gestures collected from 10 subjects, with the hand portion segmented. It was used by Gao et al. [2012] for action recognition. The 3D Iconic Gesture Dataset (3DIG)[2] contains 20 gestures performed by 29 people, each of which refer to a virtual 3D object. Iconic gestures are pictorial gestures, which depict entities, objects, or actions. Humans perform iconic gestures to refer to entities through embodying their shapes. The SKIG Dataset [Liu and Shao 2013] contains 2,160 sequences of 10 hand gestures from six subjects, with 1,080 RGB sequences and 1,080 depth sequences. All of these 10 categories are performed with three hand postures (i.e., fist, index, and flat) under three different backgrounds (i.e., wooden board, white plain paper, and paper with characters) and two illumination conditions (i.e., strong light and poor light).

[1]https://sites.google.com/a/chalearn.org/gesturechallenge/mmworkshop.
[2]http://projects.ict.usc.edu/3dig/.

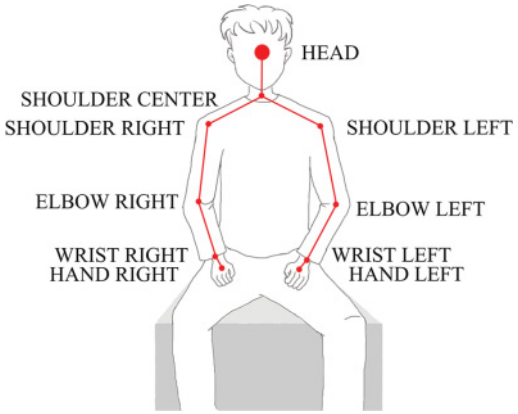

Fig. 2. Upper body skeletal joints generated using Microsoft Kinect SDK.

3. HAND DETECTION AND TRACKING

3.1. Kinect Sensor

To better detect and track hands, Kinect is employed to capture raw data streams and skeletal information in our system. It features an RGB camera and a depth sensor, which provides full-body 3D motion capture capability. Microsoft has released a Kinect Software Development Kit (SDK) for Windows, with which developers can have access to raw depth and color data streams as well as skeletal data. In our method, Kinect SDK for Windows is applied due to its stability. Kinect SDK supports the default skeletal system and seated skeletal system, and we employ the latter mode, in which 10 upper body joints (shoulders, elbows, wrists, arms, and head, both right and left, as shown in Figure 2) are tracked instead of 20 joints of the whole body. This is sufficient for our system, because sign language only contains upper body movement.

3.2. Hand Detection Algorithm

3.2.1. Skin Color Model. The use of color cues is feasible since human skin color has a predictable color distribution. We assume the signer wears long-sleeved clothes, the color of which differs from skin color.

YC_bC_r space is one of the most popular color spaces for skin color detection. The Y component corresponds to luminance, and by eliminating the Y component, two chromaticity components (C_b, C_r) constitute a color model, which is robust to illumination changes.

A probabilistic model is constructed. We assume that the C_bC_r values of skin pixels follow a bivariate Gaussian distribution $P_s(C_b, C_r)$; whether a pixel is considered to be a skin pixel can be decided by comparing its Gaussian probability to a predefined threshold. We use the signer's face to build the model online. The face detection method combines color information and depth information as well, with several differences with the hand detection method. The 3D location of the face center detected by Kinect SDK's skeletal tracking method is used directly as the face center, as the face tracking result is very robust; then, the region growing method is employed to find the face region, which is similar to the description in Section 3.2.3. The depth model resembles the hand depth model used in Section 3.2.2. The skin color model we use for face detection is to apply a threshold to C_bC_r values in Equation (1), which was found quite effective for skin color detection and very robust against different types of skin color [Chai and

(a) (b) (c)

Fig. 3. (a) Skin color detection. (b) Depth detection. (c) Hand detection from region growing.

Ngan 1999]:

$$77 \leq C_b \leq 127 \; and \; 133 \leq C_r \leq 173. \tag{1}$$

The advantage of our face detection method compared with the conventional method is that it avoids inaccuracy caused by nonskin parts in the face, such as the eyes and mouth.

The result of skin color detection is shown in Figure 3(a), where black pixels indicate non-skin-color pixels, and others are considered skin colored.

3.2.2. Depth Model. To improve the performance of hand detection, depth data captured by the Kinect depth sensor is used. In much research on 3D hand gesture recognition, a predefined depth threshold is applied to detect hands [Li and Jarvis 2009; Ren et al. 2011; Van den Bergh and Van Gool 2011]. We follow the method in Li and Jarvis [2009]. We assume the depth of a hand pixel is in a small range within the palm depth. Therefore, the hand can be found by applying a threshold,

$$d_{palm} - \Delta d_1 < z(x, y) < d_{palm} + \Delta d_2, \tag{2}$$

where d_{palm} denotes the depth of the predicted palm, (x, y) is the coordinate of a pixel, $z(x, y)$ is its depth value, and Δd_1 and Δd_2 are predefined parameters.

The depth value of each pixel is obtained from the depth image, and Kinect SDK provides the method to map the depth image onto the RGB image. The depth of the palm d_{palm} is predicted as described in Section 3.3. The result of depth segmentation is shown in Figure 3(b).

3.2.3. Region Growing. In order to detect the regions of two hands efficiently, the region-based method is employed. The advantage of the region growing method is that it does not need to go over all pixels in the picture to find the hand region.

The seed points are chosen in our hybrid tracking method; they are either the palm from skeletal tracking results and the region around it or points in the Kalman-predicted hand region that conform to the color and depth model. The details of choosing seed points are described in Section 3.3.2. The criterion to grow regions from the seed points is decided by color and depth data together, and we choose an eight-connected neighborhood to grow from seed points.

After a region is grown from a set of seed points, morphological refinement is processed. A closing operation is conducted in order to smooth the boundary of the region, and small holes in the region are removed. A region r that is larger than a predefined threshold T_{area} is considered as a detected hand r_{hand}. To find the contour of the region, we employ the border following algorithm [Suzuki et al. 1985]. Figure 3(c) shows the detected hand region after refinement is carried out, in which the blue contour and pink contour represent the left and right hands, respectively, and the yellow dot and blue dot are two detected palms.

ALGORITHM 1: Hybrid Hand Tracking

Input: Observed state in last frame $\hat{X}_{t-1}$, skeletal tracking results, and color and depth data streams
Output: Observed state $\hat{X}_t$ in this frame
1. Kalman Prediction:
   Predict the estimate of the state: $\hat{X}_t^- = A\hat{X}_{t-1}$
   Estimate error covariance: $P_t^- = AP_{t-1}A^T + Q_1$
2. Decide seed points of region growing from Kalman prediction and skeletal tracking results: $S(t) = F(S_{Kalman}(t), S_{Skeletal}(t))$
3. Detect hand region r_{hand} using region growing method
4. Decide observed palm position Z_t as the point that maximizes its distance to the boundary of r_{hand}
5. Kalman update:
   Calculate the Kalman gain: $K_t = P_t^- H^T (HP_t^- H^T + Q_2)^{-1}$
   Update the estimate: $\hat{X}_t = \hat{X}_t^- + K_t(Z_t - H\hat{X}_t^-)$
   Update the error covariance: $P_t = (I - K_t H)P_t^-$

3.3. Hybrid Hand Tracking Algorithm

Kinect SDK's skeletal tracking method provides 3D location of two palms. However, the method does not track hands well, especially when two hands are close to each other. In sign languages, hands interact with each other frequently, and many signs involve both hands held together or reaching close to each other. The region growing method we employ requires reliable palm location as a seed point; therefore, we combine the results of skeletal tracking and the Kalman tracking method for hand segmentation. The hybrid hand tracking algorithm we apply is given in Algorithm 1.

3.3.1. Kalman Parameters. We describe the state with four dimensions: $X = [s_x, s_y, v_x, v_y]'$, where (s_x, s_y) denotes the coordinate of the palm and (v_x, v_y) denotes the velocity of the palm.

For the measurement Z, we use 2D vectors: $Z = [z_x, z_y]'$, where (z_x, z_y) represents the position of the observed palm.

3.3.2. Decision of Seed Points. In the equation of the seed points decision,

$$S(t) = F(S_{Kalman}(t), S_{Skeletal}(t)), \tag{3}$$

$S(t) = \{(x, y) | (x, y) \in \vec{p}_{seed}\}$ denotes the set of predicted seed points at time t, $S_{Kalman}(t)$ denotes seed points from the Kalman prediction, $S_{Skeletal}(t)$ denotes seed points from the skeletal tracking result, and F is the decision function to choose the more reliable one to be the seed points for region growing.

$S_{Kalman}(t)$ in time t is decided from predicted Kalman state X. We assume that the hand to be tracked moves without deformation during the last frame; therefore, the hand region in time t can be predicted given the Kalman prediction of palm position and hand region in time $t-1$. Points in the predicted hand region that also conform to the color and depth model form $S_{Kalman}(t)$.

$S_{Skeletal}(t)$ consists of the palm from the skeletal tracking result and points in the 5-by-5 region around it conforming to the color and depth model.

F is a heuristic decision rule. When two hands are close enough to each other according to observation in the last frame, $S_{Kalman}(t)$ is considered more reliable. Otherwise, $S_{Skeletal}(t)$ is first selected to be the seed points. If the region grown by the selected seed points is not large enough ($Area(r) < T_{area}$), the alternative predicted set of seed points is chosen to regrow the region.

3.4. Occlusion Handling

There are two cases in which we need to tackle the occlusion problem: occlusion between two hands and occlusion between a hand and the face. We mainly employ the object hypothesis method similar to the tracking method in Argyros and Lourakis [2004] to solve the occlusion problem. In the situation when the hand is right in front of the face, Otsu's method is used to segment the hand.

3.4.1. Object Hypothesis. We assume that the spatial distribution of the pixels in a hand or face region can be approximated by an ellipse.

We use ellipse $h_i = h_i(c_{x_i}, c_{y_i}, \alpha_i, \beta_i, \theta_i)$ to model an object o_i, where (c_{x_i}, c_{y_i}) is the center of the ellipse, α_i is the major axis, β_i is the minor axis, and θ_i represents its orientation. What we need to do is to associate the object hypothesis ellipse h_i to the detected region r_j.

When occlusion doesn't occur, data association is simple, because the detected region has already been assigned to either the right or left hand in the tracking stage. We mainly discuss the hypothesis association in occlusion cases.

When occlusion occurs, two different object hypotheses h_1 and h_2 are competing for one region r_1. If a point of a detected region is within an ellipse, then the point is considered to belong to this object hypothesis. If a point of a detected region is outside any ellipse, then the point is considered to belong to the nearest object hypothesis.

After all pixels of detected region r_1 are assigned to object hypotheses, the parameters of each hypothesis are recalculated based on the set of pixels o_i assigned to it.

Finally, the object hypotheses are predicted for the next frame. The location of each object hypothesis in the next frame $(\hat{c_{x_i}}(t+1), \hat{c_{y_i}}(t+1))$ is estimated from the prediction of the respective palm, which is described in Section 3.3.2. The estimated hypothesis ellipse $\hat{h_i}(t+1) = h_i(\hat{c_{x_i}}(t+1), \hat{c_{y_i}}(t+1), \alpha_i(t), \beta_i(t), \theta_i(t))$ is used for object association in the next frame.

3.4.2. Occlusion Between Hands. When handling the situation when two hands are in touch with each other, we need to first judge whether occlusion between hands occurs. The judgment follows two rules:

—Rule 1: If two detected hand regions ($r_{rightHand}$ and $r_{leftHand}$) overlap with each other and the number of pixels that belong to both regions is larger than a threshold, then occlusion happens, and the larger one between the two detected regions ($r_{rightHand}$ and $r_{leftHand}$) is considered to be the occlusion area of two hands, $r_{bothHands}$. $r_{bothHands}$ will be used as the detected region for object hypothesis association.
—Rule 2: When two hands are close to each other according to previous observation and occlusion between the hands does not happen in the last frame, if the predicted right palm falls into the detected left hand region $r_{leftHand}$, then occlusion happens, and $r_{bothHands} = r_{leftHand}$.

Once occlusion between the hands is found, points in detected region $r_{bothHands}$ are assigned to object hypotheses of two hands, $h_{rightHand}$ and $h_{leftHand}$, as Section 3.4.1 details.

Although the occlusion region $r_{bothHands}$ is segmented into two hands approximately, it cannot represent the hands' edge accurately. Therefore, in the hand feature extraction stage, the two connected hands are not divided into right/left hand parts; instead, $r_{bothHands}$ is recorded as a hand feature (see Figure 21 in Appendix A.1). The advantage of this approach is that by taking the connected two hands as a whole, recognition of joint hands is much more accurate than recognition of either hand separately; on the other hand, by segmenting $r_{bothHands}$ into two hands, the approximate position and velocity of both hands are estimated for further tracking.

3.4.3. Occlusion Between Hand and Face. The user's face is also represented by a hypothesis ellipse. In cases where a hand overlaps with the signer' face, different approaches are processed in two situations.

When a hand is touching the face without large-area occlusion, the predicted face hypothesis h_{Face} is in the detected hand area $r_{rightHand}$ or $r_{leftHand}$, but the predicted palm is not too close to the predicted face center. The object hypothesis method described in Section 3.4.1 can handle this problem.

When a hand is right in front of the face with large-area occlusion, the predicted palm is very close to the predicted face center. In this case, the object hypothesis method cannot segment the hand accurately, and instead, Otsu's method is employed. Since the depth value of the hand and face is distinctly different in this situation, Otsu's method in depth images can automatically calculate the optimum threshold separating the face and hand.

4. HAND POSTURE RECOGNITION BASED ON DEEP NEURAL NETWORKS

Once the hands are segmented, resized, and normalized, hand segments are classified into postures using deep neural networks.

Deep neural networks are artificial neural networks that contain more than two layers of latent variables; this is also called deep learning. The goal of using deep learning is to extract and select features from input data automatically and then classify them into corresponding classes.

Deep learning has been widely used in recent years in various research fields. Hinton et al. [2006] describe a fast learning algorithm for deep belief nets (DBNs) and use it for handwritten character recognition. They show how to use complementary priors to eliminate the effects that make inference difficult in densely connected belief nets. Salakhutdinov and Hinton [2008] show how to use unlabeled data and a DBN to learn a good covariance kernel for a Gaussian process. If the data is high dimensional and highly structured, a Gaussian kernel applied to the top layer of features in the DBN works much better than a similar kernel applied to the raw input. Nair and Hinton [2009] introduce a top-level model for DBNs and evaluate it on a 3D object recognition task. Besides DBNs, Hinton and Salakhutdinov [2006] describe an effective way to initialize the weights that allows deep autoencoder networks to learn low-dimensional codes that work much better than principal components analysis as a tool to reduce the dimensionality of data. Dahl et al. [2012] propose a novel context-dependent deep neural network hidden Markov model (CD-DNN-HMM) for large-vocabulary speech recognition (LVSR). They train the DNN to produce a distribution over senones (tied triphone states) as its output.

In our work, we use DBNs for hand posture recognition. Unfortunately, DBNs' training process is difficult to parallelize across computers. Thus, we also use convolution neural networks (CNNs) for comparison. Convolution neural networks, which can be trained in parallel on GPUs or server clusters, are the earliest successful example of deep neural networks and have also been used for handwritten character recognition [LeCun et al. 1989], face detection [Nebauer 1998], and action recognition [Ji et al. 2013].

4.1. Deep Belief Networks

Deep belief networks are probabilistic generative models with many layers of hidden units above a bottom layer of visible variables that receive input data. DBNs have undirected connections between the top two layers and directed connections between the lower layers as shown in Figure 4(a). A DBN can be regarded as a stack of Restricted Boltzmann Machines (RBMs), as shown in Figure 4; every two adjacent layers are

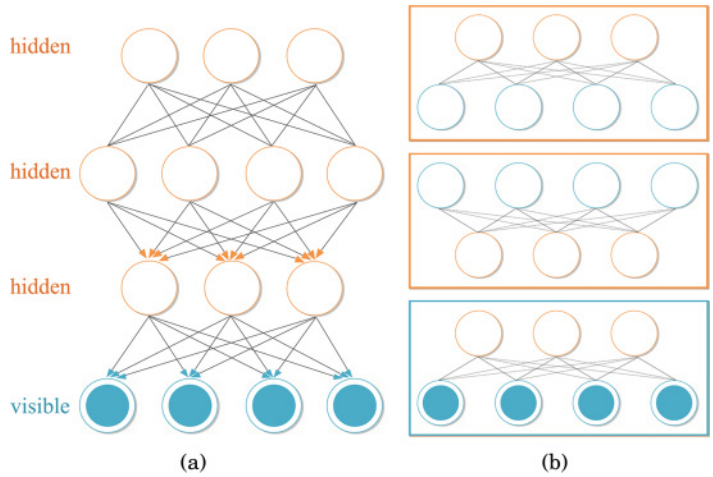

Fig. 4. (a) Deep belief network. (b) A stack of RBMs corresponding to (a).

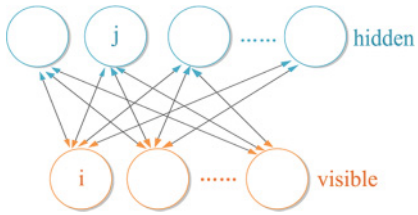

Fig. 5. Restricted Boltzmann Machine.

regarded as an RBM. The training process of a DBN can be summarized as pretraining and fine-tune; more details are described in Hinton et al. [2006].

4.1.1. Restricted Boltzmann Machines. Restricted Boltzmann Machines have one visible layer and one hidden layer. The visible and hidden units form a bipartite graph without visible–visible or hidden–hidden connection, as shown in Figure 5. Since there is no visible–visible or hidden–hidden connection, the conditional distribution over visible units $\boldsymbol{v}$ and hidden units $\boldsymbol{h}$ are given in logistic functions:

$$P(h_j = 1|\boldsymbol{v};\theta) = \sigma\left(\sum_i w_{ij}v_i + a_j\right) \tag{4}$$

$$P(v_i = 1|\boldsymbol{h};\theta) = \sigma\left(\sum_j w_{ij}h_j + b_i\right), \tag{5}$$

where $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{b}, \boldsymbol{a}\}$ are model parameters: w_{ij} represents the symmetric interaction between visible unit i and hidden unit j; b_i and a_j are bias terms; and $\sigma(x)$ is a logistic function.

The model parameters $\boldsymbol{\theta}$ can be obtained by using contrastive divergence, as described in Hinton [2002]. For the visible–hidden weight updates, we have

$$\Delta\boldsymbol{w} = -P(\boldsymbol{h} = 1|\boldsymbol{v}^{(0)})\boldsymbol{v}^{(0)} + P(\boldsymbol{h} = 1|\boldsymbol{v}^{(1)})\boldsymbol{v}^{(1)}, \tag{6}$$

ALGORITHM 2: Contrastive divergence for RBM

$\boldsymbol{w} = CDforRBM(\boldsymbol{v}^{(0)}, epoch, \epsilon)$;
Input: A sample $v^{(0)}$ from training sets, learning rate ϵ, number of iterations *epoch*.
Output: The RBM weight matrix $\boldsymbol{w}$.
Initialize $\boldsymbol{w}$ with random values from 0 to 1;
$index = 0$;
repeat
  gibbs sampling $\boldsymbol{h}^{(0)}$ using $P(\boldsymbol{h}|\boldsymbol{v}^{(0)})$;
  gibbs sampling $\boldsymbol{v}^{(1)}$ using $P(\boldsymbol{v}|\boldsymbol{h}^{(0)})$;
  $\Delta\boldsymbol{w} = -P(\boldsymbol{h} = 1|\boldsymbol{v}^{(0)})\boldsymbol{v}^{(0)} + P(\boldsymbol{h} = 1|\boldsymbol{v}^{(1)})\boldsymbol{v}^{(1)}$;
  $\boldsymbol{w} = \boldsymbol{w} + \epsilon\Delta\boldsymbol{w}$;
  $index++$;
until $index \geq epoch$;

ALGORITHM 3: Pretraining (greedy layer-by-layer algorithm) for DBN

$\boldsymbol{w} = preTrainDBN(\boldsymbol{v}^{(0)}, epoch, \epsilon, L)$;
Input: A sample $v^{(0)}$ from training sets, learning rate ϵ, number of iterations *epoch*, number of layers L except the visible layer.
Output: The weight matrix $\boldsymbol{w}^i$ of layer i, $i = 1, 2, \ldots, L-1$.
$\boldsymbol{h}^0 = \boldsymbol{v}^{(0)}$; where $\boldsymbol{h}^i$ represents the value of units in i^{th} hidden layer. $\boldsymbol{h}^0$ denotes the input layer.
$layer = 1$;
repeat
  $\boldsymbol{w}^{layer} = CDforRBM(\boldsymbol{h}^{layer-1}, epoch, \epsilon)$;
  gibbs sampling $\boldsymbol{h}^{layer}$ using $P(\boldsymbol{h}^{layer}|\boldsymbol{h}^{layer-1})$;
until $layer \geq L$;

where $\boldsymbol{v}^{(0)}$ is a sample from training sets, and $\boldsymbol{v}^{(1)}$ is a reconstructed data over one-step Gibbs sampling on $\boldsymbol{v}^{(0)}$. Finally, we can update visible–hidden weights as

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta\boldsymbol{w}. \tag{7}$$

Contrastive divergence for RBM is summarized in Algorithm 2.

4.1.2. Pretraining. Pre-training is also called the greedy layer-by-layer algorithm. Pretraining this model is to train a stack of RBMs, since every two adjacent layers can be regarded as an RBM and we use contrastive divergence to train RBM weights.

Once we have trained the first layer RBM on training data $\boldsymbol{v}$, we use Equation (4) to compute hidden unit activation probabilities $P(\boldsymbol{h}|\boldsymbol{v})$. We use $P(\boldsymbol{h}|\boldsymbol{v})$ as training data for the higher RBM, and recursively for the next layer RBMs. Thus, each set of RBM weights can be used to extract features from the output of the previous payer. When pretraining is completed, we have the initial values for all the weights of the network with the number of hidden layers the same as the number of RBMs we trained. The procedure is summarized in Algorithm 3

4.1.3. Fine-Tune. Deep belief networks are capable of extracting high-level representations from high-dimensional visible data. In order to achieve hand posture recognition, we add an output layer with randomly initialized weights at the top of the DBN after pretraining all parameters θ of the DBN, as shown in Figure 6.

Since fine-tune is supervised learning, we need to know the corresponding label for each hand image. In this step, we use back-propagation to fine-tune all the weights with labeled input images.

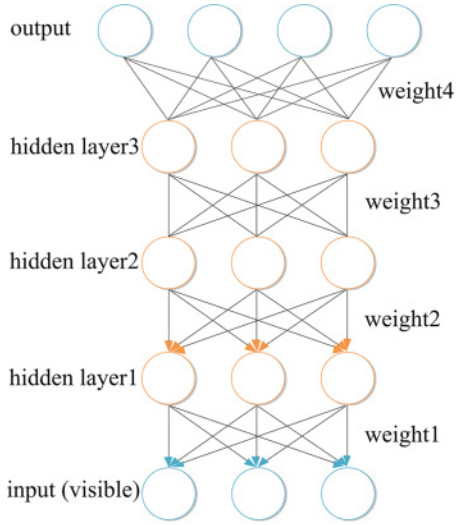

Fig. 6. DBN with an output layer.

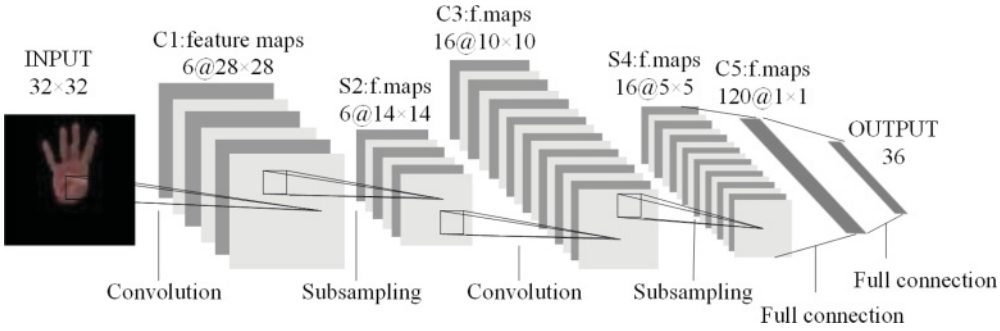

Fig. 7. Architecture of LeNet-5.

4.2. Convolution Neural Networks

CNNs are the earliest successfully established deep architecture. LeCun and Bengio [1995] designed and trained convolutional networks, obtaining state-of-the-art performance on several pattern recognition tasks. The architecture of a CNN is shown in Figure 7. More details about CNNs have been described in LeCun et al. [1998].

Each unit in a layer receives inputs from a set of units located in a small neighborhood in the previous layer. The small neighborhood is called receptive fields. Hidden layers are organized with several planes within which all the units share the same weights. The plane is called the feature map. The input layer receives images of hand shapes that are normalized. There are two kinds of hidden layers in CNNs: the convolution layer and the subsampling layer. A convolution layer is composed of several feature maps. Different feature maps can extract different types of features from the previous layer; therefore, convolution layers are used as features extractors. A subsampling layer follows a convolution layer and has the same number of feature maps as the previous convolution layer. Each feature map in subsampling layers performs averaging and subsampling on the feature map in the previous layer; therefore, a subsampling layer can reduce the resolution of the feature map. The number of units in the output layer

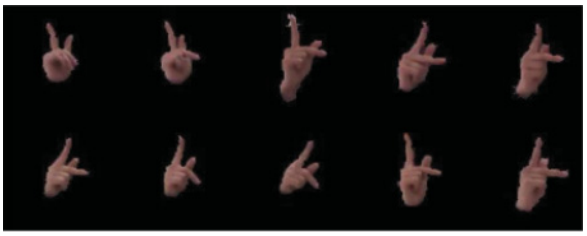

Fig. 8. Hand posture images for "K."

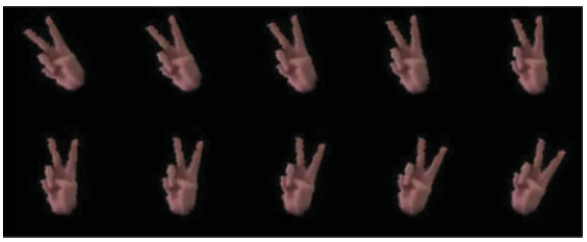

Fig. 9. Rotated images of "V."

is equal to the number of predefined classes, and each unit represents a class of hand postures.

5. EXPERIMENTS

In this section, we conducted a variety of experiments to test various aspects of the proposed system. The experiments are organized as follows:

—Hand segmentation and tracking experiments
—Posture training and recognition experiments
—Complexity experiments

In hand segmentation and tracking experiments, we test the effectiveness of our algorithms on hand segmentation and hand tracking. Our segmentation and tracking results are shown with contours of detected hands. A blue contour and a pink contour represent the left and right hands, respectively, and a yellow dot and a blue dot are two detected palms. Some representative snapshots of the experiment are demonstrated.

In posture training and recognition experiments, we consider 36 hand postures for American Sign Language (see Figure 19 and Figure 20 in Appendix A.1). We captured 36×8 video clips from eight different signers (seven males and one female) across these hand postures using our proposed segmentation methods described in Section 3. All signers are trained to perform the gestures. The frame rate of the video is 30. We extract frames from recorded video clips and collect 50,700 frame images in total. All images are converted to grayscale and resized to a resolution of 32×32 pixels. The intensity of each pixel is normalized to $(0, 1)$. Hand postures of the same class are slightly different from each other (see Figure 8). In order to achieve a robust recognition result, we randomly rotate each image from -30° to $+30^\circ$ (see Figure 9) 10 times; thus, the number of images in the database reaches $50{,}700 \times 10 = 507{,}000$. The database is divided into a training set and a testing set according to the scale 2:1. Thus, the final training set and testing set consist of 338,000 images and 169,000 images separately.

In complexity experiments, we test the training time and recognition time using DBNs and CNNs and present the runtime of our proposed system.

Fig. 10. (a) Segmentation without color model. (b) Segmentation with color model. (c) Segmentation without depth model. (d) Segmentation with depth model.

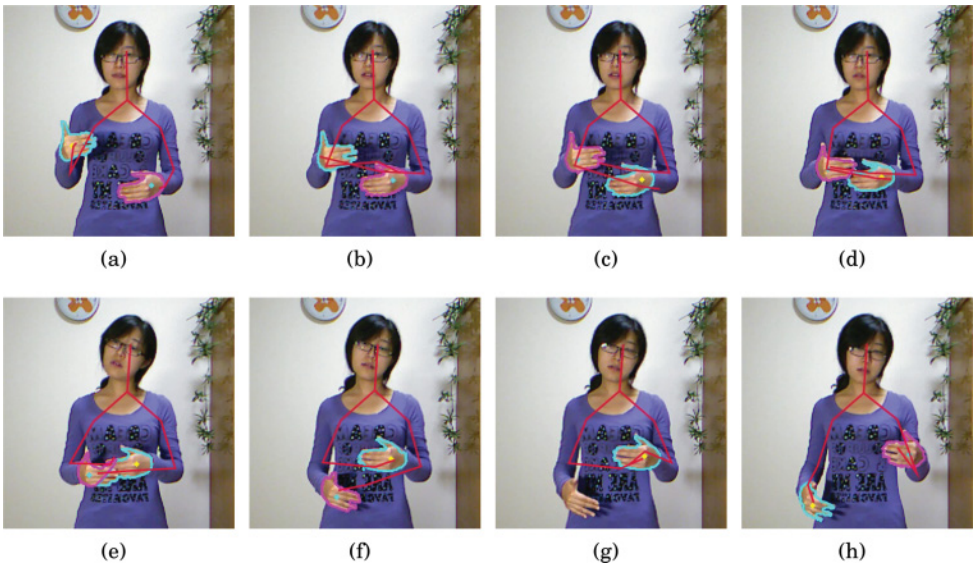

Fig. 11. Tracking results employing skeletal tracking only.

5.1. Hand Segmentation and Tracking Experiments

5.1.1. Hand Segmentation Result. The integration of the color and depth model provides stable segmentation results, and the hybrid segmentation method outperforms either single method significantly, as shown in Figure 10.

When the hand is touching other parts of the body or other objects, the depth model does not suffice to achieve successful segmentation (see Figure 10(a)), while our method can segment the hand successfully using the color model (Figure 10(b)). When the hand is in front of a skin-colored object, for instance, another man's face, Figure 10(c) illustrates the limitation of the color model, while by employing the depth model, our method still achieves satisfying results (Figure 10(d)).

However, there still remain some limitations with our segmentation method: users are supposed to wear long-sleeved shirts. By adding restrictions to the shape of the hands, we could eliminate the limitation of what users are wearing, but it is not feasible in our real-time system.

5.1.2. Hand Tracking Result. Figure 11 shows the tracking results from the skeletal tracking method provided by Kinect SDK. Predicted palms and seed points for region

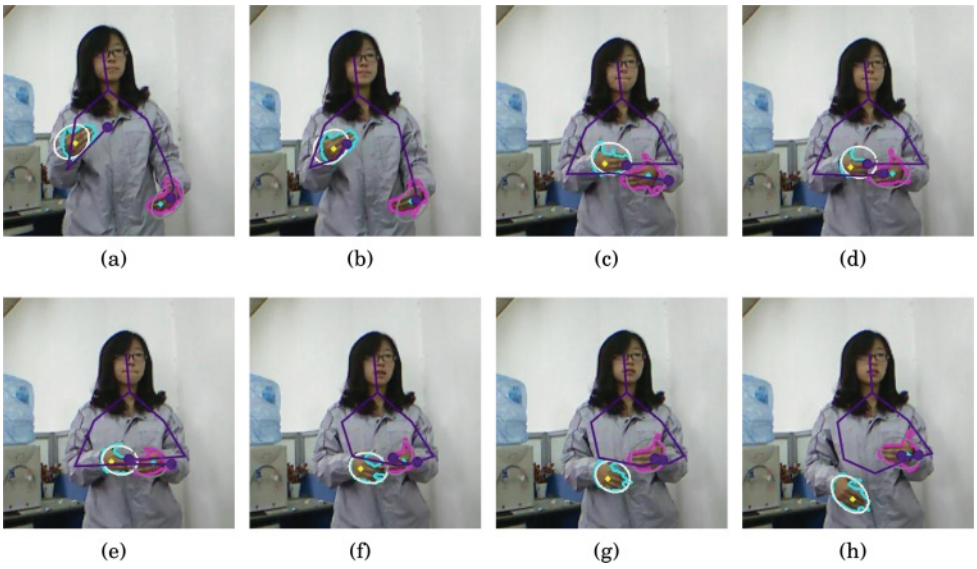

Fig. 12. Tracking results with hybrid tracking method.

growing are decided by skeletal tracking results alone. Figures 11(a) to 11(h) are eight representative frames in time sequence, and they show a movement consisting of two hands first getting close then moving apart. When two hands are close to each other, the possibility of skeletal tracking error is relatively high. In Figures 11(c) to 11(f), tracking of right and left hands is reversed because of skeletal tracking error, and in Figure 11(g), the left hand is lost. Both hands are tracked successfully again only when skeletal tracking results are corrected.

Figure 12 shows our hybrid tracking results when the signer is doing the same gesture as in Figure 11. Figures 12(a) to 12(h) are eight representative frames in time sequence. As is shown, when skeletal tracking results mistake one hand for the other, our system can track both hands successfully (Figures 12(c) to 12(e)). When skeletal tracking results lose the track of the left hand, our hybrid tracking method can track it accurately (Figures 12(f) to 12(h)).

5.1.3. Occlusion Handling. Figure 13 illustrates the hands' occlusion handling results of our system. Figures 13(a) to 13(h) are eight representative snapshots in time sequence, and they illustrate the tracking and segmentation results when two hands are getting closer until they overlap with each other, and then moving apart. Two hands are approximately segmented and successfully tracked even when hands occlude with each other severely.

When a hand is touching the face or is in front of the face, our system can segment the hand successfully. As shown in Figure 14(a), in the situation when the hand is touching the face without large-area occlusion, the face is easily mistaken as part of the hand. After we employ the object hypothesis method, the hand is segmented out of the face (see Figure 14(b)). When the hand has large-area occlusion with the face, for example, when the hand is in front of the face, as Figure 14(c) shows, it is hard to segment the hand correctly, since the depth value of the hand and face does not differ significantly. Figure 14(d) shows that Otsu's method applied in our system can handle the situation well.

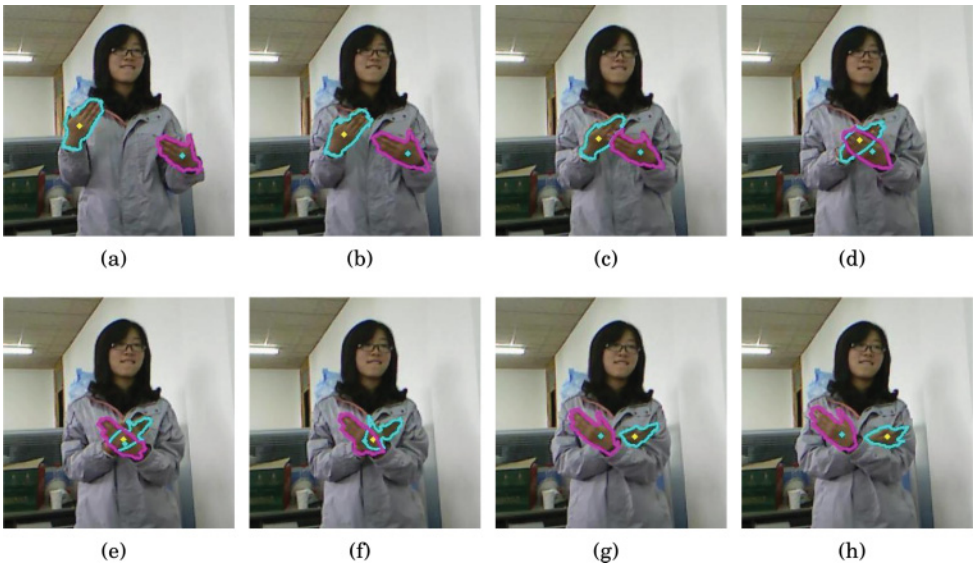

Fig. 13. Two hands overlap with each other.

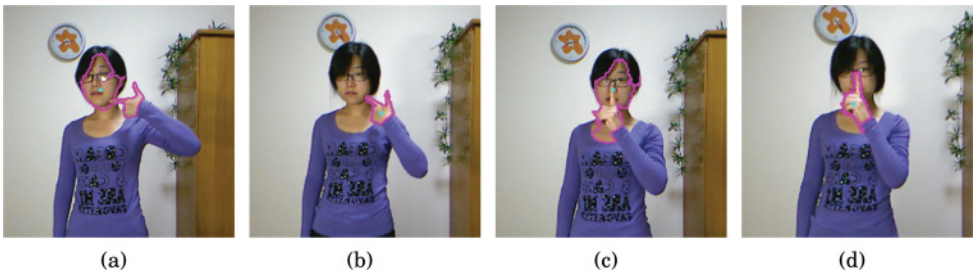

Fig. 14. Segmentation results when occlusion with face happens.

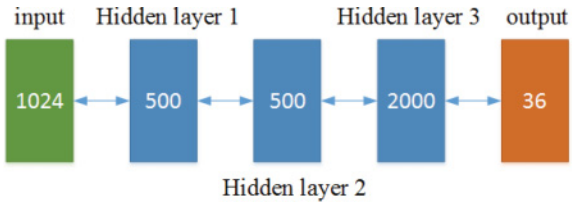

Fig. 15. Architecture of DBN used in our system.

5.2. Posture Training and Tracking Experiments

5.2.1. Training DBN. Choosing a DBN architecture with which we can obtain high accuracy within the training time we can tolerate requires experience and many trials. Hinton et al. [2006] build a three-hidden-layers model to recognize handwritten digits and obtain high accuracy. Therefore, we decide to use a three-hidden-layers DBN with 500, 500, and 2,000 hidden units in each hidden layer separately; the architecture is shown in Figure 15. There are 1,024 input units, as the resolution of the input image is 32×32 pixels. The output layer has 36 units representing the 36 classes of hand postures.

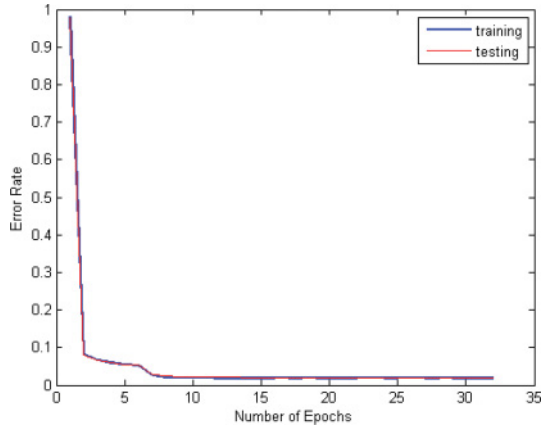

Fig. 16. Training and testing error rate of DBN as a function of the number of iterations.

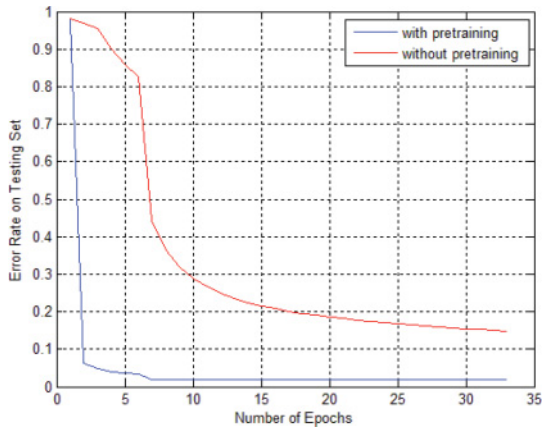

Fig. 17. The error rate on the training set while using DBN with and without pretraining.

When pretraining the first layer, intensities of all pixels are used as input units. The remaining RBMs in the stack are formed with logistic units. Training sets are used for unsupervised greedy layer-by-layer training of a DBN.

After the DBN has been pretrained on unlabeled hand posture images, a classification layer including 36 output units is fitted to the labeled images with the top layer of the DBN model as its inputs. Then we use the back-propagation algorithm to fine-tune the whole network with 32 iterations. The training and testing rate of fine-tuning are shown in Figure 16.

To validate the necessity of pretraining, we compared a DBN with and without pretraining, and the result is shown in Figure 17. From the result, we see that pretraining with unlabeled data greatly benefits the training process of a DBN.

5.2.2. Training CNN. LeCun et al. [1989] propose a convolution network called LeNet-5 for handwritten and machine-printed character recognition and get an accuracy of 99.2% trained on the MNIST database. In our work, we use LeNet-5 for hand posture recognition. The LeNet-5 comprises six layers, excluding the input. Topology of a CNN is shown in Figure 7. There are 1,024 input units, as the resolution of images is 32×32 pixels.

Table I. Parameters of CNN (LeNet-5)

Layer	Layer Type	Receptive Fields	Maps	Num. of Weights	Connections
input	input		$1 \times (32 \times 32)$		
C1	convolution	5×5	$6 \times (28 \times 28)$	156	122,304
S2	subsampling	2×2	$6 \times (14 \times 14)$	12	5,880
C3	convolution	5×5	$16 \times (10 \times 10)$	1,516	151,600
S4	subsampling	2×2	$16 \times (5 \times 5)$	32	2,000
C5	convolution	5×5	$120 \times (1 \times 1)$	48,120	48,120
output	full connection		$36 \times (1 \times 1)$	4,356	4,356

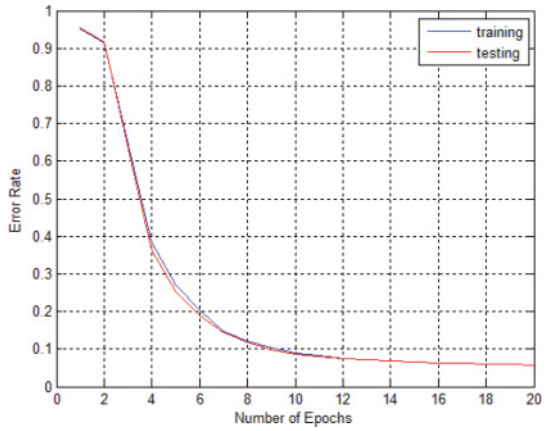

Fig. 18. Training and testing error rate of CNN as a function of the number of iterations.

The details of LeNet-5 used in our experiment are shown in Table I. For example, layer C1 is a convolutional layer with six feature maps. Each unit in each feature map is connected to a 5×5 receptive field in the input. The size of feature maps is 28×28. C1 contains 156 trainable parameters and 122,304 connections.

There are 36 output units corresponding to 36 different hand postures. It is a full connection between C5 and the output.

CNNs are trained on the training set with 20 iterations. After each iteration, we evaluate the training and testing error rate of the CNN, as Figure 18 shows.

5.2.3. Comparing Accuracies. After training, we run the DBN and CNN on a testing set. In comparison, we also test HOG+SVM on our dataset. For each image, we extract a 144-D HOG feature. The 32×32 image is divided into 16 cells. Each cell has 8×8 pixels and the gradient of a cell is divided into nine orientation bins. Every four adjacent cells constitute a block. Each image contains four blocks without overlap. Then we use LIBSVM[3] to train multiclass SVM with linear kernel for recognition. The parameter setting is following Hsu et al. [2003]. Table II shows the results of recognition accuracy of the DBN, CNN, and HOG+SVM on each hand posture.

From the recognition result, we see that both the DBN and CNN achieve a high recognition accuracy rate. The overall average of all hand postures' accuracy for the DBN is 98.12%, while the CNN is 94.17%, and HOG+SVM is 87.58%. The proposed DBN is superior to the CNN we construct, and they both get much better performance than the baseline HOG+SVM method.

[3]http://www.csie.ntu.edu.tw/∼cjlin/libsvm/.

Table II. Comparing Accuracies Between DBN and CNN

Posture	Accuracy (DBN)	Accuracy (CNN)	Accuracy (HOG+SVM)	Posture	Accuracy (DBN)	Accuracy (CNN)	Accuracy (HOG+SVM)
1	97.59%	92.61%	87.44%	**I**	98.35%	94.94%	92.32%
3	98.86%	94.92%	90.18%	**ILY**	98.47%	94.76%	91.93%
4	98.26%	94.25%	88.71%	**K**	97.97%	94.00%	90.68%
5	98.17%	93.76%	85.62%	**L**	98.21%	94.90%	89.57%
6	98.33%	94.39%	86.03%	**M**	96.78%	94.16%	84.68%
7	98.97%	95.12%	87.90%	**N**	96.96%	91.90%	81.99%
8	98.89%	93.18%	82.59%	**O**	97.84%	93.49%	87.53%
9	97.85%	92.45%	81.12%	**OpenA**	98.66%	94.77%	86.97%
A	97.49%	91.83%	84.86%	**OpenB**	98.86%	94.96%	89.29%
B	97.93%	99.71%	92.64%	**OpenM**	97.32%	93.54%	83.14%
Bent3	97.02%	92.78%	83.92%	**OpenN**	98.95%	94.85%	90.08%
BentV	96.89%	92.90%	79.98%	**OpenX**	98.67%	94.54%	92.65%
C	98.67%	94.94%	88.91%	**R**	98.36%	94.11%	86.52%
Claw5	98.45%	95.92%	91.62%	**S**	97.87%	93.16%	83.68%
Corna	98.14%	91.68%	92.58%	**T**	97.97%	94.84%	84.81%
Flat9	97.78%	92.57%	83.49%	**V**	98.64%	94.45%	90.04%
FlatB	98.28%	95.66%	86.87%	**X**	98.75%	94.49%	91.66%
FlatO	97.56%	93.24%	88.94%	**Y**	98.67%	94.73%	88.44%

Table III. Training Time of DBN

Training	Layer (1024-500) time (min)	Layer (500-500) time (min)	Layer (500-2000) time (min)
pretrain	56.2	20.5	93.4
fine-tune		315.0	
total		485.1	

Table IV. Training Time of CNN

Iteration	Time (min)	Iteration	Time (min)	Iteration	Time (min)	Iteration	Time (min)
1	39.1	**6**	38.9	**11**	38.9	**16**	39.1
2	43.4	**7**	40.5	**12**	39.0	**17**	40.8
3	39.1	**8**	38.9	**13**	39.2	**18**	39.8
4	38.9	**9**	38.8	**14**	39.2	**19**	39.5
5	38.8	**10**	39.2	**15**	39.1	**20**	40.2
total	790.4						

5.3. Complexity Experiments

5.3.1. Training Time and Recognition Time. The DBN is first pretrained by the greedy layer-by-layer algorithm, and then back-propagation is applied to fine-tune the whole network. The training time of pretraining and fine-tune is shown in Table III. The training process of a CNN is different, and the training times for each iteration are shown in Table IV.

Although it takes 485.1 minutes to train a DBN and 790.4 minutes to train a CNN on our training set, recognition on a certain hand image is in real time. As shown in Table V, the average recognition time is 0.899ms with a DBN and 1.165ms with a CNN.

For comparison, the training process of SVM needs only 15 minutes, which is much shorter than that of a DBN and CNN; the average recognition time is 15.259ms with HOG+SVM, which is a little bit longer than that of a DBN and CNN.

5.3.2. Runtime. In this section, we discuss the runtime of the proposed system. The system is implemented in C# with Microsoft Kinect SDK for Windows. On a Windows 7 computer with an Intel Core i7 3.5GHz CPU and 16GB RAM, our hand tracking

Table V. Recognition Time of DBN and CNN

model	Recognition Time on Testing Set (sec)	Average Recognition Time (sec)
DBN	152	0.000899
CNN	197	0.001165
HOG+SVM	2580	0.015259

and segmentation method takes 78.96ms for both hands on average in processing each frame. According to Table V, we see that the recognition time for two hands is around 2ms; that is, our framework is working at about 12fps, which is an acceptable result for a real-time recognition system.

6. CONCLUSIONS

In this article, we develop a robust and real-time hand posture recognition system based on a Kinect. In our system, we propose a hand detection and tracking algorithm to segment hands from the scene and apply deep neural networks to recognize postures. The proposed hand detection and tracking algorithm relies on both color and depth data and therefore is invariant to content and illumination of the scene. As a result, we obtain very good performance on hand detection and tracking and effectively overcome the occlusion problem of hands. With a deep neural network, we don't have to select specific features manually, as it can automatically detect and extract meaningful features by itself. We've implemented DBNs and CNNs for training and recognition, and although DBNs and CNNs spend hours training a model, they both work really fast on recognition. The experiment results show that our system can realize satisfactory real-time performance and high recognition accuracy. For future work, we will extend the system to be used for sign language recognition.

APPENDIX

A.1. Hand Postures for American Sign Language

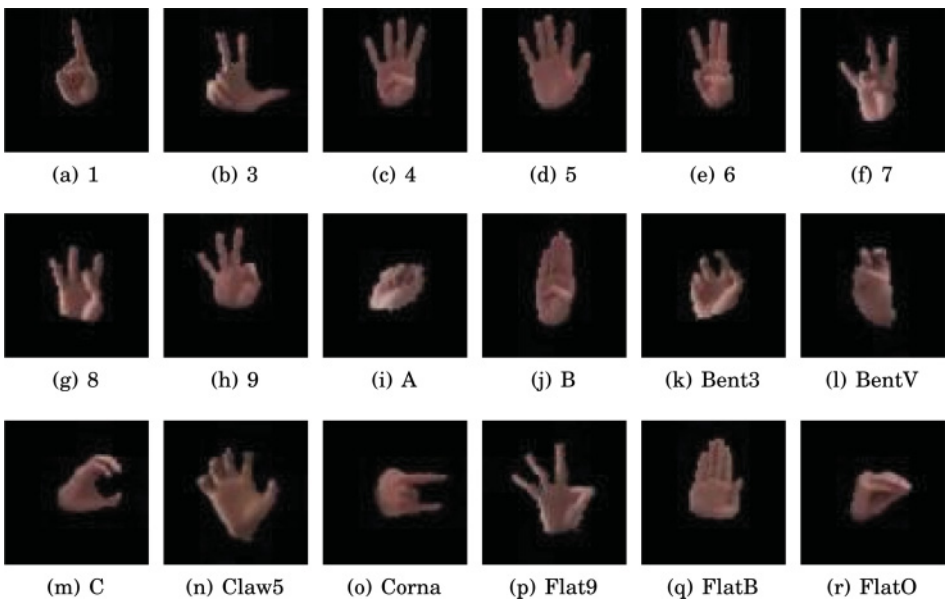

Fig. 19. Single hand postures (1).

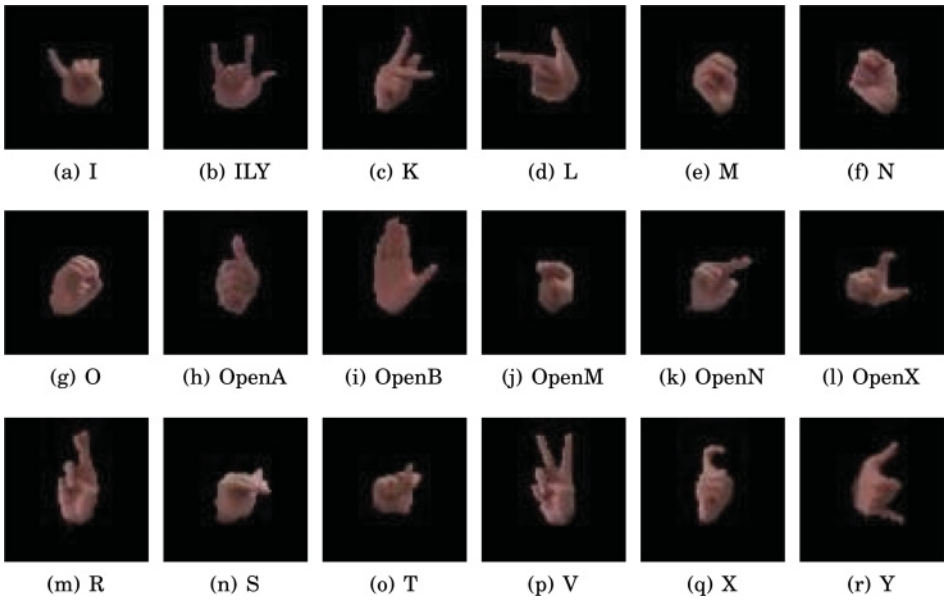

Fig. 20. Single hand postures (2).

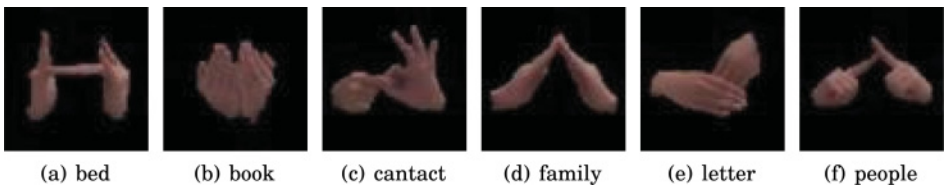

Fig. 21. Two-hand postures.

REFERENCES

Alper Aksaç, Orkun Öztürk, and Tansel Özyer. 2011. Real-time multi-objective hand posture/gesture recognition by using distance classifiers and finite state machine for virtual mouse operations. In *Proceedings of the 2011 7th International Conference on Electrical and Electronics Engineering (ELECO'11)*. IEEE, II–457.

Antonis A. Argyros and Manolis I. A. Lourakis. 2004. Real-time tracking of multiple skin-colored objects with a possibly moving camera. In *Proceedings of the European Conference on Computer Vision (ECCV'04)*. Springer, 368–379.

Chuqing Cao and Ruifeng Li. 2010. Real-time hand posture recognition using Haar-like and topological feature. In *Proceedings of the 2010 International Conference on Machine Vision and Human-Machine Interface (MVHI'10)*. IEEE, 683–687.

Manuel Caputo, Klaus Denker, Benjamin Dums, and Georg Umlauf. 2012. 3D hand gesture recognition based on sensor fusion of commodity hardware. In *Mensch & Computer 2012: interaktiv informiert–allgegenwärtig und allumfassend!?*

Douglas Chai and King N. Ngan. 1999. Face segmentation using skin-color map in videophone applications. *IEEE Transactions on Circuits and Systems for Video Technology* 9, 4 (1999), 551–564.

Feng-Sheng Chen, Chih-Ming Fu, and Chung-Lin Huang. 2003. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing* 21, 8 (2003), 745–758.

George E. Dahl, Dong Yu, Li Deng, and Alex Acero. 2012. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on Audio, Speech, and Language Processing* 20, 1 (2012), 30–42.

Marco Fagiani, Emanuele Principi, Stefano Squartini, and Francesco Piazza. 2013. A new system for automatic recognition of italian sign language. In *Neural Nets and Surroundings*. Springer, 69–79.

Gian Luca Foresti. 1999. Object recognition and tracking for remote video surveillance. *IEEE Transactions on Circuits and Systems for Video Technology* 9, 7 (1999), 1045–1062.

Wen Gao, Gaolin Fang, Debin Zhao, and Yiqiang Chen. 2004. A Chinese sign language recognition system based on SOFM/SRN/HMM. *Pattern Recognition* 37, 12 (2004), 2389–2402.

Geoffrey E. Hinton. 2002. Training products of experts by minimizing contrastive divergence. *Neural Computation* 14, 8 (2002), 1771–1800.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation* 18, 7 (2006), 1527–1554.

Geoffrey E. Hinton and Ruslan R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507.

Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. 2003. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University. July, 2003.

Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 2013. 3D convolutional neural networks for human action recognition. (2013).

Alex Krizhevsky and Geoffrey E. Hinton. 2011. Using very deep autoencoders for content-based image retrieval. In *Proceeding of the European Symposium on Artificial Neural Networks (ESANN'11).*

A. Kurakin, Z. Zhang, and Z. Liu. 2012. A real time system for dynamic hand gesture recognition with a depth sensor. In *Proceedings of the 20th European Signal Processing Conference (EUSIPCO'12)*. IEEE, 1975–1979.

Yann LeCun. 1989. Generalization and network design strategies. *Connectionism in Perspective* (1989), 143–155.

Yann LeCun and Yoshua Bengio. 1995. Convolutional networks for images, speech, and time series. *The Handbook of Brain Theory and Neural Networks* 3361, 310 (1995).

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.

Billy Y. L. Li, Ajmal S. Mian, Wanquan Liu, and Aneesh Krishna. 2013. Using Kinect for face recognition under varying poses, expressions, illumination and disguise. In *Proceeding of the 2013 IEEE Workshop on Applications of Computer Vision (WACV)*. IEEE, 186–192.

Yi Li. 2012. Hand gesture recognition using Kinect. In *Proceedings of the 2012 IEEE 3rd International Conference on Software Engineering and Service Science (ICSESS)*. IEEE, 196–199.

Zhi Li and Ray Jarvis. 2009. Real time hand gesture recognition using a range camera. In *Proceedings of the Australasian Conference on Robotics and Automation*. 21–27.

Li Liu and Ling Shao. 2013. Learning discriminative representations from RGB-D video data. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

S. Malassiotis and M. G. Strintzis. 2008. Real-time hand posture recognition using range data. *Image and Vision Computing* 26, 7 (2008), 1027–1037.

Vinod Nair and Geoffrey Hinton. 2009. 3-d object recognition with deep belief nets. *Advances in Neural Information Processing Systems* 22 (2009), 1339–1347.

C. Nebauer. 1998. Evaluation of convolutional neural networks for visual recognition. *IEEE Transactions on Neural Networks* 9, 4 (1998), 685–696.

V. Radha and M. Krishnaveni. 2009. Threshold based segmentation using median filter for sign language recognition system. In *Proceedings of the World Congress on Nature & Biologically Inspired Computing, 2009 (NaBIC'09).* IEEE, 1394–1399.

Zhou Ren, Junsong Yuan, and Zhengyou Zhang. 2011. Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. In *Proceedings of the 19th ACM International Conference on Multimedia*. ACM, 1093–1096.

Ruslan Salakhutdinov and Geoffrey E. Hinton. 2008. Using deep belief nets to learn covariance kernels for Gaussian processes. *Advances in Neural Information Processing Systems* (2008), 1249–1256.

Frank Seide, Gang Li, and Dong Yu. 2011. Conversational speech transcription using context-dependent deep neural networks. In *Proceedings of Interspeech*. 437–440.

Poonam Suryanarayan, Anbumani Subramanian, and Dinesh Mandalapu. 2010. Dynamic hand pose recognition using depth data. In *Proceedings of the 2010 20th International Conference on Pattern Recognition (ICPR'10)*. IEEE, 3105–3108.

Satoshi Suzuki. 1985. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing* 30, 1 (1985), 32–46.

Balazs Tusor and A. R. Varkonyi-Koczy. 2010. Circular fuzzy neural network based hand gesture and posture modeling. In *Proceedings of the 2010 IEEE Instrumentation and Measurement Technology Conference (I2MTC'10)*. IEEE, 815–820.

Michael Van den Bergh and Luc Van Gool. 2011. Combining RGB and ToF cameras for real-time 3D hand gesture interaction. In *Proceedings of the 2011 IEEE Workshop on Applications of Computer Vision (WACV'11)*. IEEE, 66–72.

Jiang Wang, Zicheng Liu, Jan Chorowski, Zhuoyuan Chen, and Ying Wu. 2012. Robust 3D action recognition with random occupancy patterns. In *Proceedings of the European Conference on Computer Vision (ECCV'12)*. Springer, 872–885.

Yue Gao, Meng Wang, Dacheng Tao, Rongrong Ji, and Qh Dai. 2012. 3D object retrieval and recognition with hypergraph analysis. *IEEE Transactions on Image Processing* 21, 9 (2012), 4290–4303.

Meng Wang, Xian-Sheng Hua, Tao Mei, Richang Hong, Guojun Qi, Yan Song, and Li-Rong Dai. 2009. Semi-supervised kernel density estimation for video annotation. *Computer Vision and Image Understanding* 113, 3 (2009), 384–396.

Lu Xia, Chia-Chih Chen, and J. K. Aggarwal. 2012. View invariant human action recognition using histograms of 3D joints. In *Proceedings of the 2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW'12)*. IEEE, 20–27.

X. Zabulis, H. Baltzakis, and A. Argyros. 2009. Vision-based hand gesture recognition for human-computer interaction. *The Universal Access Handbook*. LEA (2009).

Received July 2013; revised November 2013; accepted January 2014