Neural Comput & Applic (2017) 28:3941–3951
DOI 10.1007/s00521-016-2294-8

ORIGINAL ARTICLE

Deep learning in vision-based static hand gesture recognition

Oyebade K. Oyedotun[1] · **Adnan Khashman**[1,2]

Received: 13 September 2015 / Accepted: 30 March 2016 / Published online: 11 April 2016

Abstract Hand gesture for communication has proven effective for humans, and active research is ongoing in replicating the same success in computer vision systems. Human–computer interaction can be significantly improved from advances in systems that are capable of recognizing different hand gestures. In contrast to many earlier works, which consider the recognition of significantly differentiable hand gestures, and therefore often selecting a few gestures from the American Sign Language (ASL) for recognition, we propose applying deep learning to the problem of hand gesture recognition for the whole 24 hand gestures obtained from the Thomas Moeslund's gesture recognition database. We show that more biologically inspired and deep neural networks such as convolutional neural network and stacked denoising autoencoder are capable of learning the complex hand gesture classification task with lower error rates. The considered networks are trained and tested on data obtained from the above-mentioned public database; results comparison is then made against earlier works in which only small subsets of the ASL hand gestures are considered for recognition.

Keywords Hand gesture recognition · Human–computer interaction · Neural network · Deep learning

✉ Oyebade K. Oyedotun
oyebade.oyedotun@ecraa.com

Adnan Khashman
adnan.khashman@ecraa.com

[1] European Centre for Research and Academic Affairs (ECRAA), Lefkosa, Mersin-10, Northern Cyprus, Turkey

[2] University of Kyrenia, Karakum, Kyrenia, Mersin-10, Northern Cyprus, Turkey

1 Introduction

Humans are very apt in recognizing hand gestures, with which further decisions can be made. After a period of learning, we are able to almost effortlessly communicate using hand gestures. In recent times, significant research has been ongoing in developing machines which are capable of classifying captured images of hand gestures into the considered categories. Such systems have found applications in game and electronic (TV, DVD, etc.) control, robot control, virtual reality environments, and natural language communication [1, 2]. In many works, the recognition of static hand gestures is based on significantly or relatively differentiable gestures, since it is obvious that some of the hand gestures are quite similar when viewed in 2D. Hence, subsets of hand gesture signs found in databases are extracted for recognition task. Nevertheless, small distinguishing features are present in all the gestures, which can be perceived by humans. The capability of human–computer interaction systems to extend its vocabulary by being capable of recognizing more hand gestures is very important, as this allows an expansion in the level of communication and control.

Some approaches to the problem of hand gesture recognition include local orientation histograms, support vector machine (SVM), artificial neural network (ANN) and elastic graph matching (EGM) [3–6]. Support vector machine can be seen as a maximum margin classifier, in which support vectors are used to determine the hyperplane which gives maximum separation of the hand gesture classes [7]. Also, backpropagation neural network has been used to learn features extracted from hand gesture images [8]. However, in vision-based recognition systems, processed images have been used to train the networks (or classifiers) [9, 10]. Generally, neural networks have found

machine learning applications in many other diverse areas such as face recognition, blood cell identification and prediction of students' performance [11–13].

Deep learning, a relatively recent approach to machine learning, which involves neural networks with more than one hidden layer, has been used with much success in face recognition, speech recognition and natural language processing tasks [14–16]. Networks based on deep learning paradigms enjoy more biologically inspired architectures and learning algorithms, in contrast to conventional feed-forward networks. Generally, deep networks are trained in a layer-wise fashion and rely on more distributed and hierarchical learning of features as it is found in the human visual cortex [17]; these allow the representation of highly nonlinear functions, discovery of more interesting features in training data and better modeling of complex problems.

In this work, we apply deep learning-based networks such as convolutional neural network (CNN) and stacked denoising autoencoder (SDAE) to the task of recognizing 24 American Sign Language (ASL) hand gestures obtained from a public database [18].

2 Deep learning

Advances in understanding human visual processing have shown that perception is achieved in a hierarchical fashion. Biological motivation for deep learning stems from studies on the visual cortex which suggests that abstract features from visual data are combined into primary features in the second layer; these features are further combined into more defined features in the next layer; these features are then further combined into more interesting features in the following layers [19, 20]. Attempts at modeling problems with deep networks have been met with the difficulty in training these networks using the conventional backpropagation learning algorithm; many researches show that performances actually degrade as we stacked more hidden layers [21]. These poor performances have been shown to be due to saturating neurons and vanishing gradients [22], while over-fitting is as a result of increased number of neurons as more hidden layers are stacked. However, much success has been recorded with the relatively recent training of deep networks in a greedy layer-wise fashion, in which deep network weights are learned one layer at a time in a phase referred to as pretraining (generative model). For a classification task (discriminative model), the whole network can be fine-tuned using the backpropagation learning algorithm [23]. The following subsections of this section briefly describe convolutional neural network (CNN) and stacked denoising autoencoder (SDAE).

2.1 Convolutional neural network (CNN)

Generally, convolutional neural networks are composed of convolution layers, sub-sampling layers and a classifier layer (e.g., SVM or multilayer network). Each convolution layer is followed by its corresponding sub-sampling layer. The convolution layer and sub-sampling layers have feature maps that are arranged in 2D topology; several layers can be stacked in these networks to realize a deep network. Figure 1 shows a typical convolutional neural network. Note that our network topology lumps a convolution and its sub-sampling layer together as a layer. Convolutional networks rely on the following unique attributes briefly discussed below [24, 25].

- Local receptive field: Each neuron is connected to only a region in the input.
- Weight sharing: Neurons in the same feature map have the same weights.
- Sub-sampling: It reduces the spatial dimension of convolution feature maps and therefore required number of neurons. Also, it introduces some invariance.

2.1.1 Convolution operation

From Fig. 1, the input image is assumed to be of size $K \times L$, and a kernel of size $a \times a$ used for the convolution operation, which generates n convolution feature maps (C1: first convolution map) of size $i \times i$ each. Furthermore, it is assumed that a kernel of size $a \times a$ is again used to convolve sub-sampling layer S1 (first sub-sampling layer) to obtain the second convolution layer C2, with m feature maps of size $d \times d$ each.

2.1.2 Sub-sampling/pooling operation

It is in this layer that extracted features in each convolution feature map are spatially reduced. From Fig. 1, S1 (first sub-sampling layer) is obtained by max pooling feature maps in C1 using a mask size of $b \times b$. Also, to obtain S2 (second sub-sampling layer), it is assumed that the same mask size is used to pool feature maps C2.

2.2 Stacked denoising autoencoder (SDAE)

Autoencoders are generative networks, which can be used to learn the reconstruction of input data at the output layer (pretraining). These networks use an unsupervised learning algorithm, since class labels are not used for training. Generally, an autoencoder can be considered as having one hidden layer, which learns the compressed representation

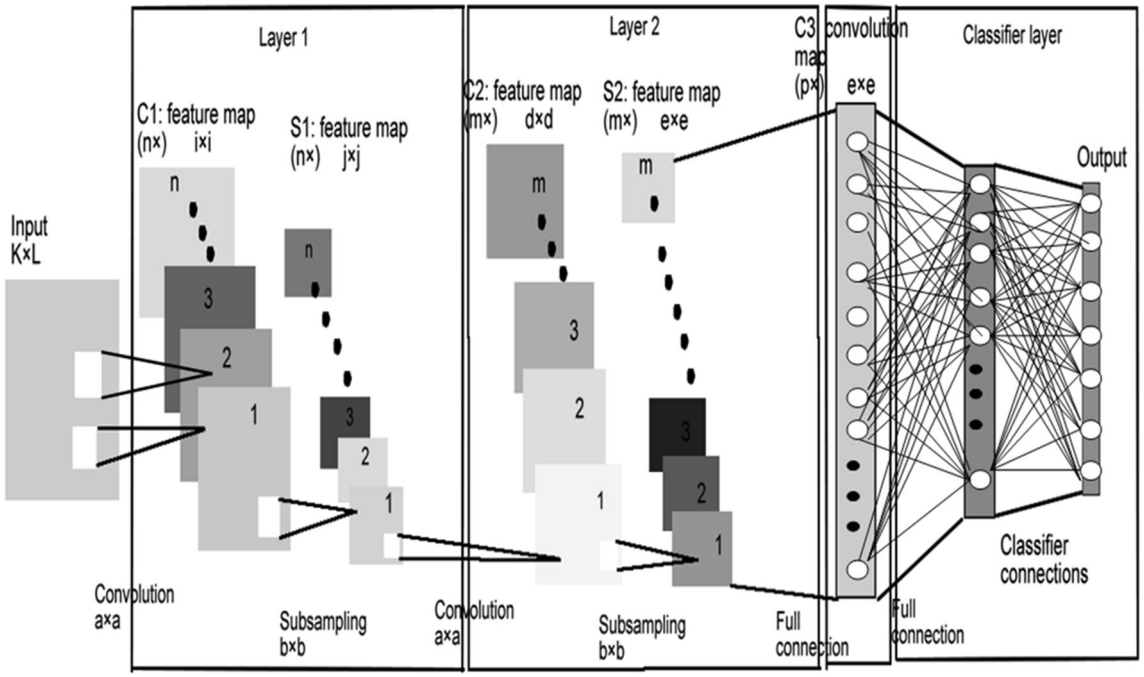

Fig. 1 Convolutional neural network [24]

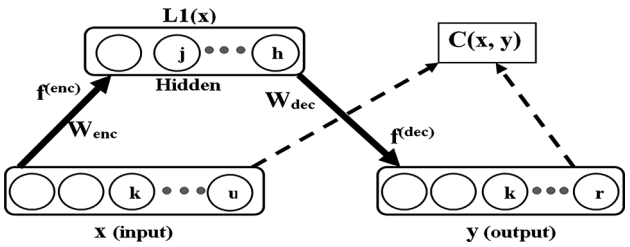

Fig. 2 Autoencoder

of input attributes (i.e., as in encoding function: $f^{(\text{enc})}$), and its expansion to the original input data in the output layer (i.e., as in decoding function: $f^{(\text{dec})}$). Figure 2 shows a typical autoencoder, where x is the input data, L1(x) a vector of hidden layer activations and y is the computed output. The encoding stage is the learning of activations of hidden neurons, while the decoding stage is the learning of the output neuron activations from the already learned hidden layer neuron activations. Input data are supplied at the input layer, and the same input data are supplied at the output layer as the desired output; the encoder weights of the network (W_{enc}), and decoder weights (W_{dec}) are then learned by optimizing a cost function. For binary-valued inputs, the sum of Bernoulli cross-entropies can be used as the cost function as described in Eq. 1 [26]. Note that for autoencoders, $u = r$, i.e., Fig. 1.

$$C(x, y) = -\sum_{k=1}^{r} (x_k \log(y_k) + (1 - x_k) \log(1 - y_k)) \qquad (1)$$

where x_k is the kth attribute in the input data vector x, and y_k is the corresponding kth attribute of the computed output data vector, y. The loss function is minimized toward $x_k = y_k$, i.e., when the input is correctly reconstructed in the output layer.

A variant of the typical autoencoder is the denoising autoencoder (DAE). This network is used to learn the reconstruction of corrupted input data in the output layer. The input data are corrupted to a particular degree, usually 50 %, while the desired output data are the uncorrupted data. It has been shown that such networks learn more interesting features from training data compared to the conventional autoencoder [27].

For classification tasks, the autoencoder can be fine-tuned with the backpropagation learning algorithm using the class labels of the training data. Furthermore, it has been shown that by stacking more hidden layers, more distributed and hierarchical learning of features is achieved [27].

3 Data analysis

The data used in this work are obtained from a public database [18] and contain 24 different hand gestures. Figure 3 shows the 24 unprocessed static hand gestures for recognition. It will be seen that the hand gestures are signed against uniformly lit dark background. Also, it is observed that alphabets j and z are missing from the public

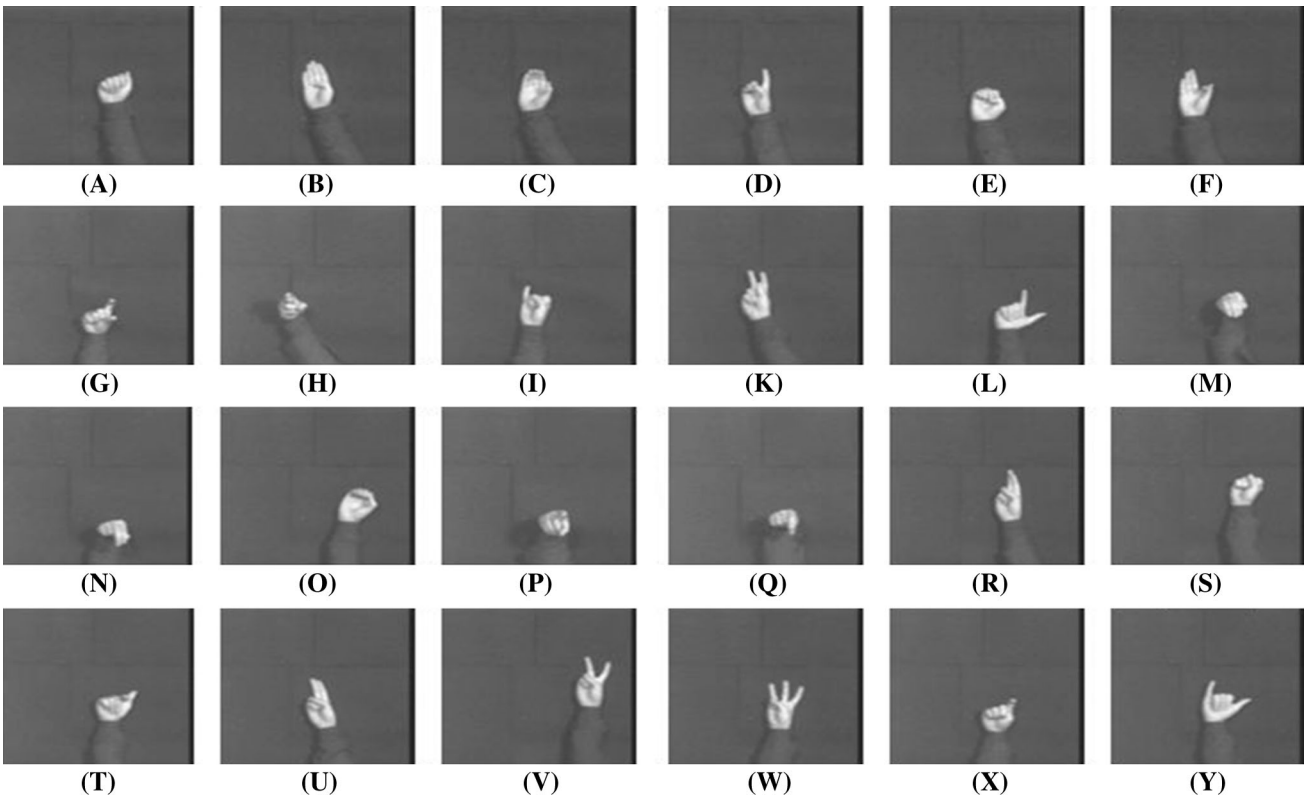

Fig. 3 Twenty-four unprocessed ASL static hand gestures

database, since they are nonstatic. In order not to over-bias learning, 1440 hand gestures (~70 %) and 600 hand gestures (~30 %) are used for training and testing the designed networks, respectively. The image samples of training and testing data contained in the obtained database are given in Fig. 3.

3.1 Data preprocessing and segmentation

The original hand gestures images are of size 248 × 256 pixels in grayscale; these images are transformed to binary (black and white) in order to prepare them for segmentation. The binary images are obtained by thresholding the original images at 0.5 gray level; the threshold value was found suitable for segmenting hands as white pixels (1 s) from the black background with pixel values of 0. Furthermore, the obtained binary images are filtered using a median filter of size 15 × 10 in order to remove noise which may affect the hand segmentation stage, as original images are not 100 % noise free. The operations described above can be considered as image preprocessing operations. Figure 4 shows samples of unsegmented binary hand gestures.

An algorithm to segment hand-occupied regions in the binary images is then implemented. The hand segmentation algorithm tracks the boundary of the white pixels in the images, and extracts pixels in form of a rectangular bounding box containing the segmented hand. The extracted images are then rescaled to 32 × 32 pixels using the pattern averaging technique; this reduces computational requirements and training time of networks. Figure 5 shows samples of the 24 segmented hand gestures considered in this work.

The flowchart for the proposed recognition system in this work is shown in Fig. 6. Note that loop 1 shows the training loop of the system, while loop 2 shows the testing loop of the system.

4 Training of networks

The considered networks in this work are trained on data described in Sect. 3. Training data comprise 1440 processed hand gesture images of size 32 × 32 pixels each. The following subsections describe the training of CNN and SDAE. The number of stacked layers is varied for both networks in order to investigate obtainable performances on the recognition task. All networks are trained and simulated on a dual core, intel (R) Pentium (R) (2.00 GHz) CPU with 3 GB RAM, with programs written and run in a MATLAB environment. Also, note that all networks in this work are trained with the aim of obtaining the hyper-parameters which yield best performances of networks; it is

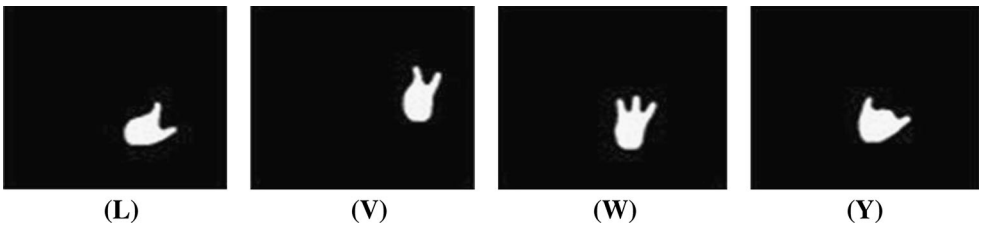

Fig. 4 Unsegmented hand gesture samples

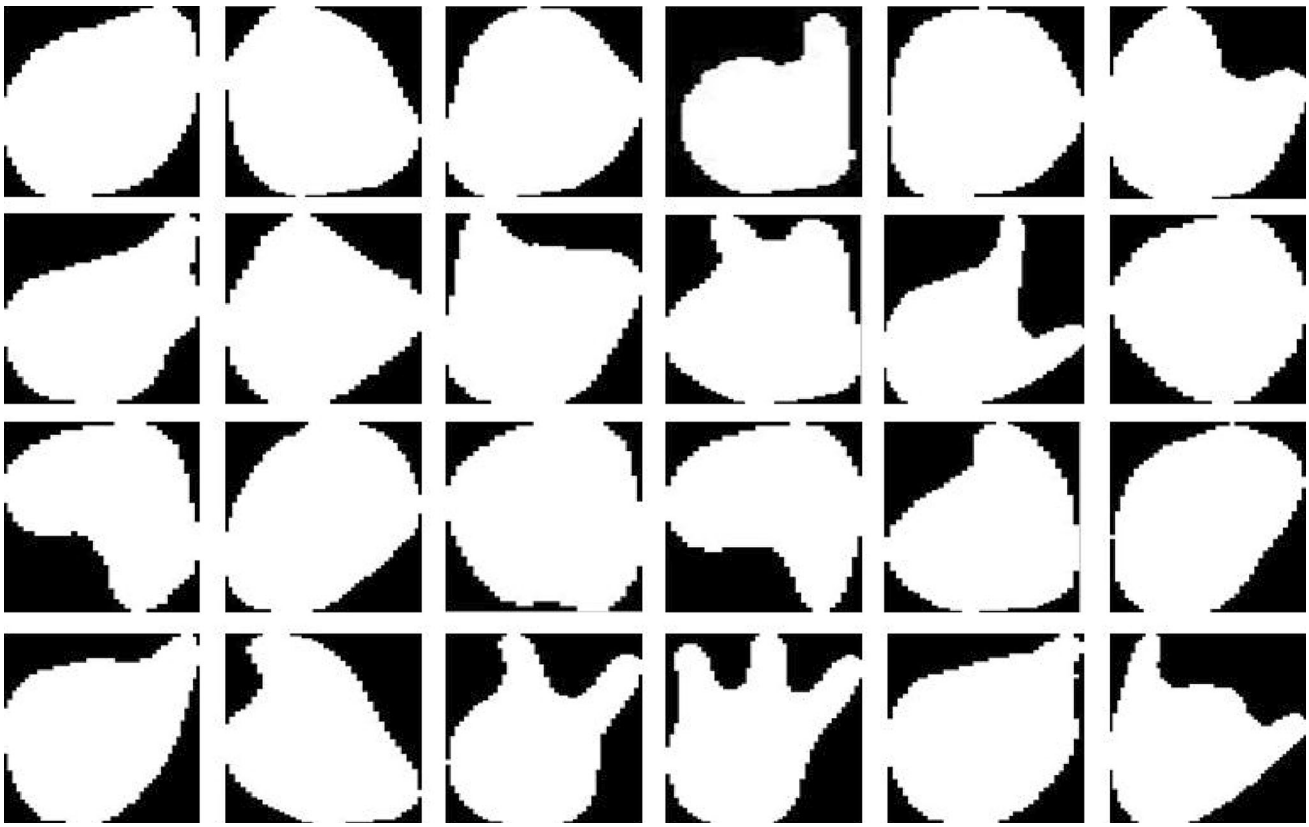

Fig. 5 Segmented hand gestures

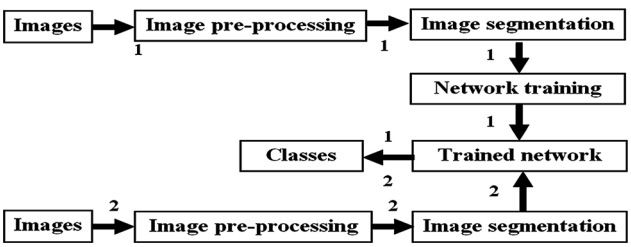

Fig. 6 Flowchart of the proposed system

the major aim of this work to report the observed maximum obtainable performances of the considered networks as can be found in some other related works [28–31].

4.1 CNN training

Convolutional neural networks (CNNs) of different depth sizes are trained in this work. The architecture of the three CNNs: CNN1, CNN2 and CNN3 are described below. Convolution kernels of various and suitable sizes are used in the networks as obtains in some other related works [30, 32].

4.1.1 CNN1

First, a CNN of two hidden layers, CNN1, is trained. The architecture of CNN1 is described below.

- First hidden layer (H1): A kernel of size 5×5 is used for the convolution operation, and six convolution feature maps of size 28×28 each are generated. Furthermore, a pooling window of size 2×2 is used for the sub-sampling operation, and six sub-sampling feature maps of size 14×14 each are generated.
- Second hidden layer (H2): A kernel of size 5×5 is used for convolution operation, and 12 feature maps of size 10×10 each are generated. Also, a pooling window of size 2×2 is used for sub-sampling operation, and 12 sub-sampling feature maps of size 5×5 each are generated.

Furthermore, the classifier layer is a fully connected network of one hidden layer with 400 neurons. Considering the number of hand gesture classes, it therefore follows that the classifier output layer has 24 neurons. A batch size of 5 is used to achieve stochastic gradients computations for optimizing the mean-square-error cost function. Final training parameters for CNN1 are shown in Table 1. The learning curve for CNN 1 is shown in Fig. 7.

4.1.2 CNN2

Furthermore, we experiment with a CNN of three hidden layers. In order to achieve this, we rescale the training data images to 64×64 pixels; this is necessary to achieve a deeper network, since pooling operations successively reduce spatial dimensions of data. The architecture of CNN2 is described below.

- First hidden layer (H1): A kernel of size 5×5 is used for the convolution operation, and six convolution feature maps of size 60×60 each are generated. Furthermore, a pooling window of size 2×2 is used for the sub-sampling operation, and six sub-sampling feature maps of size 30×30 each are generated.
- Second hidden layer (H2): A kernel of size 5×5 is used for the convolution operation, and 12 feature maps of size 26×26 each are generated. Also, a pooling window of size 2×2 is used for the sub-sampling operation, and 12 sub-sampling feature maps of size 13×13 each are generated.
- Third hidden layer (H3): A kernel of size 5×5 is used for the convolution operation, and 12 feature maps of size 9×9 each are generated. Also, a pooling window of size 3×3 is used for the sub-sampling operation, and 12 sub-sampling feature maps of size 3×3 each are generated.

Where H3 is the third hidden layer for the CNN; the classifier layer remains as described for CNN1. The training parameters for CNN2 are shown in Table 1; a batch size of 5 is used to achieve stochastic training of the network. The learning curve for CNN2 is shown in Fig. 7. It can be seen that training required more training time and the achieved MSE is higher as compared to that of CNN1.

4.1.3 CNN3

Also, CNN3 with four hidden layers is trained, as this allows the observation of improvements in learning with network depth; CNN3's architecture is described below.

- First hidden layer (H1): A kernel of size 3×3 is used for the convolution operation, and five convolution feature maps of size 62×62 each are generated. Furthermore, a pooling window of size 2×2 is used for the sub-sampling operation, and five sub-sampling feature maps of size 31×31 each are generated.
- Second hidden layer (H2): A kernel of size 4×4 is used for the convolution operation, and ten feature maps of size 28×28 each are generated. Also, a pooling window of size 2×2 is used for the sub-sampling operation, and ten sub-sampling feature maps of size 14×14 each are generated.
- Third hidden layer (H3): A kernel of size 3×3 is used for the convolution operation, and 15 feature maps of size 12×12 each are generated. Also, a pooling window of size 2×2 is used for the sub-sampling operation, and 15 sub-sampling feature maps of size 6×6 each are generated.
- Fourth hidden layer (H4): A kernel of size 3×3 is used for the convolution operation, and 20 feature maps of size 4×4 each are generated. Also, a pooling window of size 2×2 is used for the sub-sampling

Table 1 CNN training parameters

Network	CNN1	CNN2	CNN3
Number of training samples	1440	1440	1440
Activation function	Log-Sigmoid	Log-Sigmoid	Log-Sigmoid
Learning rate (η)	0.8	0.8	0.8
Iterations	14,400	14,400	20,000
Training time (s)	523	620	745
Mean square error (MSE)	0.0812	0.0989	0.1021

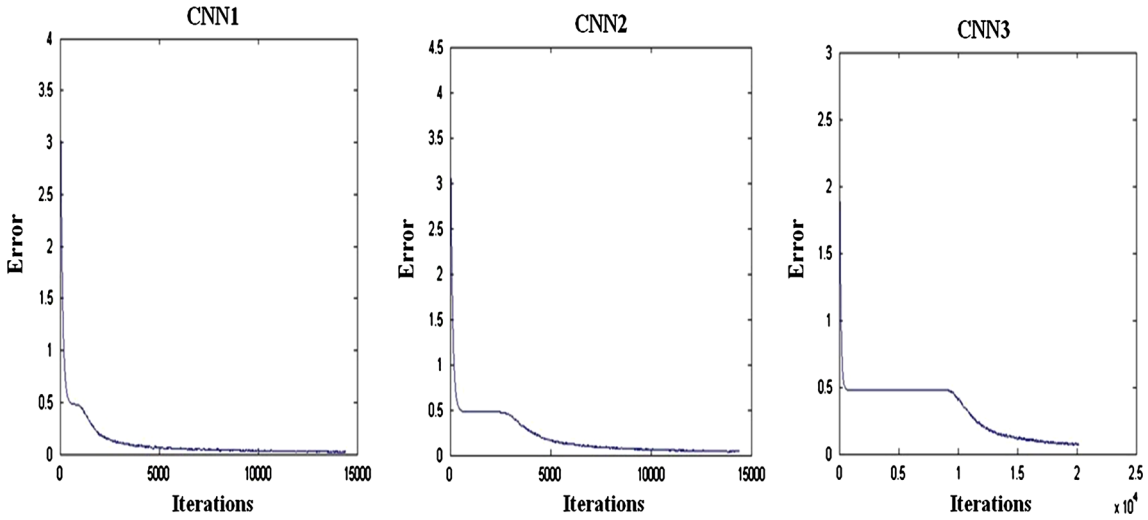

Fig. 7 Learning curves for CNNs

operation, and 20 sub-sampling feature maps of size 2×2 each are generated.

The classifier layer for CNN3 is a fully connected network of one hidden layer with 400 neurons as in CNN1 and CNN2, and a batch size of 5 is used to achieve stochastic training of the network. The final training parameters for CNN3 are shown in Table 1. The learning curve for CNN3 is shown in Fig. 7. It can be seen from Table 1 that as the depth of CNN is increased, it becomes more difficult to train, i.e., CNN3 has a higher MSE compared to CNN2 and CNN1. Moreover, training required most number of iterations.

4.2 SDAE training

Stacked denoising autoencoders (SDAEs) of different depths are trained in order to observe any improvement in performance as more layers are stacked. The three different SDAEs are denoted SDAE1, SDAE2 and SDAE3. All SDAEs have 1024 input neurons (since input data are of size 32×32 pixels) and 24 output neurons, i.e., number of hand gesture classes.

4.2.1 SDAE1

The architecture of SDAE1 is shown in Table 2. From Table 2, it can be seen that SDAE1 has 90 and 70 neurons in the first and second hidden layers of the network, respectively. Learning rates of 0.2 and 0.4 are used for greedy layer-wise pretraining and fine-tuning, respectively, and a momentum rate of 0.5 is used as inertia for the error gradient (it improves the possibility of escaping poor local minima). A mean square error (MSE) of 0.0310 is achieved after fine-tuning.

4.2.2 SDAE2

The architecture for SDAE2 with three hidden layers is shown below in Table 2. It will be seen that SDAE2 achieved a lower MSE compared to SDAE1.

4.2.3 SDAE3

Lastly, a SDAE3 with four hidden layers is trained; the network architecture and training parameters are given in Table 2. It can be seen that SDAE3 achieved the lowest MSE out of the three trained SDAEs.

4.2.4 Learned kernels for stacked denoising autoencoders

The learned kernels by neurons in the first hidden layer of SDAE1, SDAE2 and SDAE3 are shown in Fig. 8. It will be seen that the neurons in the first layer of SDAE1 are relatively active on features contained in the training data. However, compared to SDAE1, neurons in the first layer of SDAE2 have improved activity as can be observed from its learned kernels. Lastly, it can be seen from the learned kernels in SDAE3 that its first hidden layer neurons are the most active (learned more interesting representations) as compared to neurons in the first layers of SDAE1 and SDAE2. Generally, networks which learn more interesting features tend to have better performances at run time, as good prior knowledge (hidden representations) acquired in the unsupervised pretraining phase is important for fine-tuning and classification.

Table 2 SDAE training parameters

Network	SDAE1	SDAE2	SDAE3
Number of training samples	1440	1440	1440
Number of hidden layer 1 neurons	90	90	120
Number of hidden layer 2 neurons	70	70	90
Number of hidden layer 3 neurons	–	40	50
Number of hidden layer 4 neurons	–	–	40
Activation function	Log-Sigmoid	Log-Sigmoid	Log-Sigmoid
Learning rate for pretraining ($\eta 1$)	0.2	0.2	0.8
Learning rate for Fine-tuning ($\eta 2$)	0.4	0.4	0.4
Momentum rate for Fine-tuning (β)	0.5	0.5	0.5
Denoising ratio	0.5	0.5	0.5
Pretraining iterations	5	5	10
Fine-tuning iterations	200	200	300
Training time (s)	71	104	158
Mean square error (MSE)	0.0310	0.0120	0.0098

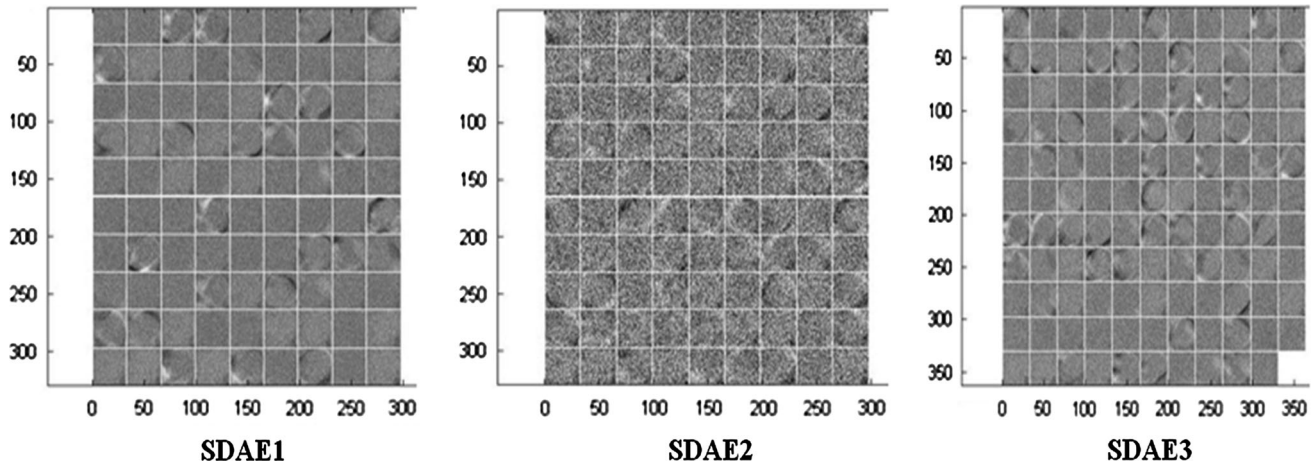

Fig. 8 Learned kernels for the stacked denoising autoencoders (SDAEs)

5 Network testing and discussion

The trained convolutional neural networks (CNNs) and stacked denoising autoencoders (SDAEs) are first tested with the training data, 1440 hand gestures. In order to obtain the generalization power of the trained networks, 600 samples of hand gestures that are not part of the training data are used to test the networks. Table 3 shows the performance parameters of networks on the training data. The performance of the networks described in this work are expressed based on recognition rates (r.r) calculated using Eq. 2.

$$\text{r.r} = \frac{\text{Number of correctly classified samples}}{\text{Total number of test samples}} \quad (2)$$

It will be seen that CNN1 with two hidden layers achieved the highest recognition rate for the trained CNNs. Also, it will be seen that there is decline in recognition rates for CNN2 and CNN3; this may be attributed to saturating neurons and vanishing gradients, i.e., with the depth of the

Table 3 Performances of networks on training data

Parameters	CNN1	CNN2	CNN3	SDAE1	SDAE2	SDAE3
Number of testing samples	1440	1440	1440	1400	1400	1400
Correctly simulated samples	1413	1387	1347	1404	1407	1432
Recognition rate (%)	98.13	96.32	93.54	97.50	97.71	99.44
Run time (s)	0.83	0.97	1.32	0.49	0.55	0.67

CNNs. Furthermore, it can be seen that as the depth of the CNNs is increased, the run time for simulation increases.

In contrast, it is observed that recognition rate increases as more layers are stacked in the denoising autoencoders, i.e., SDAE3 with four hidden layers achieved the highest recognition rate from the trained SDAEs. However, it can be seen that the run time for the SDAEs increases with depth.

Table 4 shows the performances of trained networks on testing data. It will be seen that the recognition rates for the CNNs decrease with networks' depth, i.e., CNN3 has the lowest recognition rate. In contrast, recognition rates for the SDAEs increase with networks' depth, i.e., SDAE3 achieved highest recognition rate. Also, during the testing of networks, it is observed that most errors in classification of hand gestures occurred for similar hand gestures. The three most misclassified hand gestures are shown in Fig. 9.

From Fig. 9, it is shown that hand gestures C and E are misclassified for each other; hand gestures D and G are misclassified for each other; and hand gestures G and T are misclassified for each other. This error is conceivable considering the similarity of these most misclassified hand gestures, since the hand gestures are better appreciated in 3D or by applying advanced image processing algorithms to them.

Table 5 shows the comparison of the designed networks in this work with some earlier works (EW), EW1, EW2 and EW3. It will be seen that the proposed systems within this work (in bold) achieved higher recognition rates in identifying the 24 ASL hand gestures, as compared to systems in other works which are of lower recognition rates in classifying 24, 10 and 6 hand gestures. Only works which state explicitly achieved recognition rates on the test data are considered for comparison. Our results show that applying deep learning to the problem of vision-based static hand gesture recognition is promising, in that more similar hand gestures can be recognized with higher recognition rates, in contrast to many other works which select subsets of significantly distinguishable hand gestures in databases for recognition. Deep learning-based neural networks benefit from more distributed and hierarchical learning which have been shown to contribute to their superior performances in many applications, and therefore in this work.

Table 4 Performances of networks on testing data

Parameters	CNN1	CNN2	CNN3	SDAE1	SDAE2	SDAE3
Number of testing samples	600	600	600	600	600	600
Correctly simulated samples	548	534	503	539	551	557
Recognition rate (%)	91.33	89.00	83.83	89.83	91.83	92.83
Run time (s)	0.57	0.78	1.05	0.32	0.41	0.59

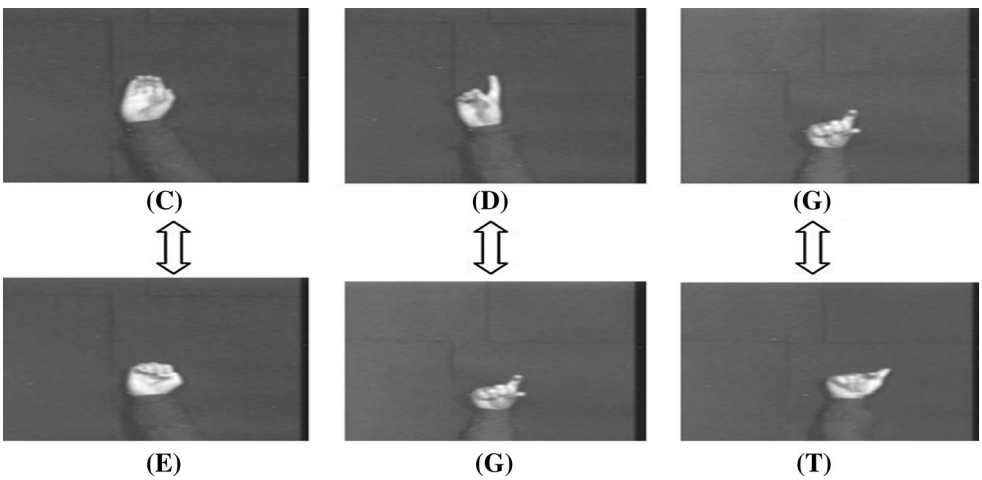

Fig. 9 Most misclassified hand gestures

Table 5 Results comparison with earlier works

Parameters	CNN1	SDAE3	EW1 [33]	EW2 [34]	EW3 [35]
Number of gestures for recognition	24	24	24	10	6
Recognition rate (%)	91.33	92.83	86.3	83.31	86.38

6 Conclusion

Hand gesture recognition systems are very important in human–computer interaction environments. The capability of such systems to reasonably recognize different hand gestures with which further crucial decisions can be made have found applications in virtual environments, game and electronics control, robot manipulation, etc.

In contrast to many other works, in which the obvious challenge of recognizing similar hand gestures poses a major challenge; therefore, subsets of hand gestures which are relatively distinguishable are considered for recognition; deep learning has been applied to the recognition of 24 ASL (American Sign Language) hand gestures. Convolutional neural networks (CNNs) and stacked denoising autoencoders (SDAEs) are trained on a public database; recognition rates of 91.33 and 92.83 % are obtained, respectively, using test data that are not part of the training data. The authors believe that the difficulty in learning observed in the CNNs with increased depth can be overcome by using rectified linear activations in the hidden layers of the network, as this reduces the effect of neurons saturation and therefore vanishing gradients. Also, the CNNs described within this work should benefit from pretraining schemes which initialize the network weights; this has been shown effective in optimizing and regularizing deep networks.

The capability of machines to effectively process a large number of different hand gestures while running with low error rates is one major source of challenge in hand gesture recognition. For this situation, we show that the systems described within this work are robust enough to learn 24 different static hand gestures with lower error rates, as against recognition systems described in other works where few different hand gestures are selected for recognition. This work is important in that robust hand gesture recognition systems which can cope with a large number of different hand gestures are crucial for the expansion of knowledge database in human–computer interaction systems. Moreover, this allows for more control and communication.

References

1. Nguyen T-N, Huynh H-H, Meunier J (2013) Static hand gesture recognition using artificial neural network. J Image Graph 1(1): 34–38
2. Nagi J, Ducatelle F, Di Caro GA et al (2011) Max-pooling convolutional neural networks for vision-based hand gesture recognition. In: 2011 IEEE international conference on signal and image processing applications (ICSIPA2011), pp 342–347
3. Rahman MdH, Afrin J (2013) Hand gesture recognition using multiclass support vector machine. Int J Comput Appl 74(1):39–43
4. Sultana A, Rajapuspha T (2012) Vision based gesture recognition for alphabetical hand gestures using the SVM classifier. Int J Comput Sci Eng Technol 3(7):218–223
5. Yewale SK, Bharne PK (2011) Hand gesture recognition using different algorithms based on artificial neural network. In: 2011 International conference on emerging trends in networks and computer communications (ETNCC), 22–24 April 2011, Udaipur, pp 287–292
6. Triesch J, von Malsburg C (2011) A system for person-independent hand posture recognition against complex backgrounds. IEEE Trans Pattern Anal Mach Intell 23(12):1449–1453
7. Oyedotun OK, Olaniyi EO, Helwan A, Khashman A (2014) Decision support models for iris nevus diagnosis considering potential malignancy. Int J Sci Eng Res 5(12):419–426
8. Ahmed T (2012) A neural network based real time hand gesture recognition system. Int J Comput Appl 59(4):17–22
9. Phu JJ, Tay YH (2006) Computer vision based hand gesture recognition using artificial neural network. Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman, pp 1–6
10. Ibraheem NA, Khan RZ (2012) Vision based gesture recognition using neural networks approaches: a review. Int J Hum Comput Interact 3(1):1–14
11. Khashman A (2012) Investigation of different neural models for blood cell type identification. Neural Comput Appl 21(6):1177–1183
12. Khashman A (2009) Application of an emotional neural network to facial recognition. Neural Comput Appl 18(4):309–320
13. Oyedotun OK, Tackie SN, Olaniyi EO, Khashman A (2015) Data mining of students' performance: Turkish students as a case study. Int J Intell Syst Appl 7(9):20–27
14. Wang W, Yang J, Xiao J et al (2015) Face recognition based on deep learning. Lect Notes Comput Sci 8944:812–820
15. Noda K, Yamaguchi Y, Nakadai K et al (2015) Audio-visual speech recognition using deep learning. Appl Intell 42(4):722–737
16. Collobert R, Weston J, Bottou L et al (2011) Natural language processing (almost) from scratch. J Mach Learn Res 12:2493–2537
17. Kruger N et al (2013) Deep hierarchies in the primate visual cortex: What can we learn for computer vision? IEEE Trans Pattern Anal Mach Intell 35(8):1847–1871
18. Thomas Moeslund's gesture recognition database—PRIMA. http://www-prima.inrialpes.fr/FGnet/data/12-MoeslundGesture/database.html
19. Najafabadi MM et al (2015) Deep learning applications and challenges in big data analytics. J Big Data 2(1):1–21
20. Pierre B (2012) Autoencoders, unsupervised learning, and deep architectures. Workshop Unsuperv Transf Learn 27:37–50
21. Erhan D, Bengio Y, Courville A (2010) Why does unsupervised pre-training help deep learning? J Mach Learn Res 11:625–660
22. Glorot X, Bengio Y (2010) Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of 13th international conference on artificial intelligence and statistics, pp 249–256
23. Hinton GE, Osindero S, Teh YW (2006) A fast learning algorithm for deep belief nets. Neural Comput 18(7):1527–1554
24. Oyedotun OK, Dimililer K (2016) Pattern recognition: invariance learning in convolutional auto encoder network. Int J Image Graph Signal Process 8(3):19–27
25. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324

26. Oyedotun OK, Olaniyi EO, Khashman A (2015) Deep learning in character recognition considering pattern invariance constraints. Int J Intell Syst Appl 7(7):1–10
27. Vincent P et al (2010) Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res 11:3371–3408
28. Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
29. Zeiler MD, Fergus R (2014) Visualizing and understanding convolutional networks. In: Fleet D, Pajdla T, Schiele B, Tuytelaars T (eds) Computer vision—ECCV 2014. Springer, Berlin, pp 818–833
30. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: Pereira F, Burges CJC, Bottou L, Weinberger KQ (eds) Advances in neural information processing systems. Neural Information Processing Systems (NIPS), pp 1097–1105
31. Sutskever I, Martens J, Dahl G, Hinton G (2013) On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th international conference on machine learning (ICML-13), pp 1139–1147
32. Scherer D, Müller A, Behnke S (2010) Evaluation of pooling operations in convolutional architectures for object recognition. In: Diamantaras KI, Duch W, Iliadis LS (eds) Artificial neural networks—ICANN. Springer, Berlin, pp 92–101
33. Hasan H, Abdul-Kareem S (2014) Static hand gesture recognition using neural networks. Artif Intell Rev 41(2):147–181
34. Avraam M (2014) Static gesture recognition combining graph and appearance features. Int J Adv Res Artif Intell 3(2):1–4
35. Nguyen T-N, Huynh H-H, Meunier J (2015) Static hand gesture recognition using principal component analysis combined with artificial neural network. J Autom Control Eng 3(1):40–45