Neural Computing and Applications (2020) 32:3135–3147
https://doi.org/10.1007/s00521-019-04152-6

ORIGINAL ARTICLE

Intrusion detection using deep sparse auto-encoder and self-taught learning

Aqsa Saeed Qureshi[1] · Asifullah Khan[1,2] · Nauman Shamim[1] · Muhammad Hanif Durad[1]

Received: 17 November 2017 / Accepted: 15 March 2019 / Published online: 26 March 2019

Abstract
With the enormous increase in the use of the Internet, secure transfer of data across networks has become a challenging task. Attackers are in continuous search of getting information from network traffic, and this is the main reason that efficient intrusion detection techniques are required to identify different kinds of network attacks. In past, various supervised and semi-supervised methods have been developed for intrusion detection. Most of these methods require a large amount of data to develop an efficient intrusion detection system. In the proposed deep neural network and adaptive self-taught-based transfer learning technique, we have exploited the concept of self-taught learning to train deep neural networks for reliable network intrusion detection. In the proposed method, a pre-trained network on regression-related task is used to extract features from NSL-KDD dataset. Original features along with extracted features from the pre-trained network are then provided as an input to the sparse auto-encoder. Self-taught learning-based extracted features, when concatenated with the original features of NSL-KDD dataset, enhance the performance of the sparse auto-encoder. Performance of self-taught learning-based approach is compared against several techniques using ten independent runs in terms of accuracy, false alarm and detection rate, area under the ROC, and PR curve. It is experimentally observed that the auto-encoder trained on the combined original and extracted features is stable and offers good generalization in comparison with the sparse auto-encoder trained on original features alone.

Keywords Self-taught learning · Deep sparse auto-encoder · Intrusion detection system

1 Introduction

In the last decade, with the rapid growth in information technology, security has become one of the major concerns of almost all the institutes and companies. Intruders are in continuous search to access the network traffic, and that is why, security systems have to cope with different types of attacks [1]. Secure data transfer over a network has always been a challenging task. In some cases, such as banking transactions, defense-related communication, traffic of government-related networks, the security of network traffic becomes increasingly important. Consequently, it is imperative to have a secure network monitoring and detection system that can efficiently detect different types of attacks and thus secure a network against malicious attacks. An intrusion detection system (IDS) monitors and detects violations regarding security policy and recognizes any abnormal patterns in the network traffic [2]. In host-based IDS, only information related to host is monitored [3]. In case of network-based IDS, instead of monitoring individual hosts in the network, an overall network flow is monitored and analyzed [4]. However, in application-based IDS, a system monitors any abnormal behavior of protocol between the different devices [5].

An IDS can be considered as an expert system. The system is first trained over a large dataset for selecting features. After training and recognizing patterns as normal or malicious, the working of IDS is tested on unseen data. The performance of an IDS mostly depends on the selected

✉ Asifullah Khan
asif@pieas.edu.pk

1 PR-Lab, Department of Computer and Information Science, Pakistan Institute of Engineering and the Applied Sciences, Nilore, Islamabad 45650, Pakistan

2 Centre for Mathematical Sciences, Pakistan Institute of Engineering and Applied Sciences, Nilore, Islamabad 45650, Pakistan

features. Intrusion detection techniques are broadly categorized into two main groups. The first one is called anomaly-based IDS, in which those data patterns are detected, which show abnormal behavior [6]. The advantage of anomaly-based IDS is that zero-day attacks can be detected; however, it may have a high false positive rate. The second category of IDS is signature- or misuse-based IDS, in which patterns are known in advance. For example, a labeled intrusion detection dataset has instances, which are labeled as normal or intrusive. By using machine learning algorithms, a trained model can automatically detect intrusive instances [7]. However, zero-day attacks are difficult to be detected by signature-based IDS.

In past, many different intrusion detection techniques have been reported by different researchers. Malhotra et al. [8] used K-nearest neighbor and genetic programming for developing a reliable IDS. To overcome the deficiencies of IDS associated with single-level structure technique, a hierarchical approach based on neural networks is recommended by Zhan et al. [9]. On the other hand, Panda et al. [10] used naïve Bayes classifier for developing an efficient IDS. Zhang et al. [11] used random forest-based classifier for the purpose of detecting an intrusion within the network. Portnoy et al. [12] proposed a clustering-based technique that uses unlabeled data for detecting an intrusion within the network. Similarly, an IDS trained in an unsupervised way was reported by Leung et al. [13]. Aslahi et al. [14] reported a hybrid approach for intrusion detection system. Another network-based IDS was developed by Ramakrishnan et al. [15] in which features are selected on the basis of entropy. Similarly, a technique that detects intrusion in wireless ad hoc networks was proposed by Srinivasan et al. [16], whereby Kohen's self-organizing map identifies abnormal patterns within the network. In contrast, Puri et al. [17] proposed a hybrid technique that combines regression and classification trees along with support vector machine (SVM) for identifying network attacks. In a different work, in order to ensure network security, SVM and neural network-based IDS are developed by Mukkamala et al. [18].

Recently, a fuzzy logic-based semi-supervised algorithm has been proposed for intrusion detection, in which a single-layer neural network is used to obtain fuzzy membership function [19]. Similarly, Kim et al. [20] used deep neural network-based approach for IDS. Another tree algorithm-based approach for network intrusion detection was reported by Subasi et al. [21]. Almost all of these intrusion detection techniques use machine learning algorithms that need to be trained from scratch. In contrast, the proposed deep neural network and adaptive self-taught-based transfer learning (DST-TL) technique uses the concept of self-taught learning for developing efficient and reliable IDS. Instead of training the model with original features as in previously reported methods, the proposed DST-TL technique uses a self-taught learning approach for extracting feature through an already trained network (using unlabeled data). The extracted features are then combined with the original features and are used to train a sparse auto-encoder. The organization of the paper is as follows: Sect. 2 is related to the description of the dataset. Similarly, Sect. 3 presents the proposed technique. Implementation and parameter details are discussed in Sect. 4. Results and discussion are described in Sect. 5. Section 6 is related to conclusions.

2 Details of dataset

In 1998, MIT Lincoln Laboratory generated an environment in which raw data from TCP dump were gathered throw local area network (LAN) that simulates Air Force LAN in the USA [22]. Each TCP connection within the network is labeled either as attacked or a normal. Moreover, training and test data contain five and two million records, respectively. In 1999, a version of the data collected from MIT Lincoln Laboratory is used in KDD contest related to intrusion detection. The purpose of gathering the data was to analyze and evaluate the research related to detecting intrusion within the network. The performance of proposed DST-TL is evaluated on NSL-KDD dataset (an updated version of KDD dataset). KDD dataset is among the few publically available network-based IDS datasets. There are some inherent deficiencies [23] in KDD dataset; some of which are:

(a) Almost 78% and 75% of the samples are repeated in training and test sets, respectively.
(b) Redundant samples in training set may make the classifier biased toward the most frequent samples; as a result, good performance may not be achieved on the test data.

To overcome the deficiencies of KDD dataset, NSL-KDD dataset was introduced by Tavallaee et al. [23]. All the redundant samples in KDD dataset were removed, and only a single copy was kept in training and test sets. Moreover, in both training and test sets, difficulty level against each record of KDD data was also assigned. For that purpose, the training set was divided into three equal parts. After the division of training data, seven different classifiers were trained on different portions of the data; as a result, 21 different labels were predicted against test data records. From KDD dataset, after the removal of redundant records $KDDTest^{+}$ was generated. In addition to $KDDTest^{+}$, a new test set $KDDTest^{-21}$ was also generated containing only those records, which were not correctly classified by any of the trained classifiers. For experimental purposes, only 20% of the $KDDTrain^{+}$ was reserved. Therefore, performance of a classifier needs to be evaluated on $KDDTest^{+}$, especially on $KDDTest^{-21}$. Percentages of the positive and negative samples in $KDDTest^{+}$ and $KDDTest^{-21}$ are shown in Table 1. In NSL-KDD dataset, each sample consists of 41 features, which are further categorized into three different groups.

1. *Basic features* The TCP/IP connection-related attributes are included in basic features. The basic features are originated from the packet header, and out of 41 features in NSL-KDD dataset, ten features are categorized as basic features. For example, protocol type, service, flags, etc. are considered as basic features.
2. *Content features* Content features are related to all of the suspicious information within the payload of a TCP packet. For example, the number of failed login efforts is considered as a content feature.
3. *Traffic features* Traffic features are calculated with reference to the time interval and are further categorized into two groups:

   (a) *Same service features* Same service features are features that are evaluated across only those connections from past 2 s that contain the same service as current connections hold.

   (b) *Same host features* Same host features are features (behavior, services of protocol, etc.) that are evaluated across those connections in past 2 s, which contain the same host as current connections hold.

Table 2 shows details of the 41 features present within NSL-KDD dataset.

All the attacks in the NSL-KDD data belong to the following four attack categories:

- Denial-of-service (DOS) attacks
- User-to-root (U2R) attacks
- Remote-to-local (R2L) attacks
- Probing (PROB) attacks

Details related to all of the four categories of attacks are shown in Table 3. Another property of the NSL-KDD dataset is that an additional 14 new attacks also are included in the test data.

3 Deep sparse auto-encoder

Sparsity is used for reducing the links within the network and thus generally increases the generalization performance of the machine learning technique. In past, different researchers proposed the sparse representation-based classification system for recognizing face [24, 25], object categorization [26], classification [27], and regression-related [28, 29] tasks. Lu et al. [24] proposed locality-based weighted sparse representation classifier (WSRC) for the face recognition-related task. Performance comparison of Lu's technique is checked on 15 different datasets from UCI repository. Experimental results depict the WSRC-based method outperforms most of the state-of-the-art techniques. Work related to recognizing the partially occluded faces within images is proposed by Mi et al. [25]. In Mi's work after the division of images into blocks, the regression-based technique is used to find out the occluded blocks. In the end, sparse representation classification (SRC)-based strategy is used to detect the face. Comparison of Mi's technique against extended SRC (eSRC) shows that SRC technique is good not only in terms of accuracy but also computationally less extensive. In another researcher, Gui et al. [30] presented a survey related to sparsity-based feature selection strategies. In Gui's survey, mathematical representation and motivation behind the sparsity-based feature selection strategies have been discussed. Another work related to group sparse multiview patch alignment framework (GSM-PAF) method for the purpose of selecting and extracting feature by introducing sparsity within the projection matrix is proposed [27]. GSM-PAF technique shows good performance on real image classification tasks. In the proposed DST-TL work, sparse auto-encoder-based self-taught learning approach is used for the development of a reliable IDS.

An auto-encoder is a kind of neural network that tries to reproduce input to its output. Two steps are involved in the training of deep sparse auto-encoder. In the first step, unsupervised greedy layer-wise pre-training is performed, in which unlabeled data are used to extract useful features at each layer. Encoding and decoding are important steps involved in the pre-training of deep sparse auto-encoder. An auto-encoder tries to reproduce its input to its output, during the pre-training phase, and that is why, labeled data are not needed during pre-training phase. Let d_{inp} be the

Table 1 Percentage of positive and negative samples in $KDDTest^{+}$ and $KDDTest^{-21}$ dataset

Dataset	Total samples	Normal samples	Abnormal samples
20% of $KDDTrain^{+}$	25,193	13,449 (53.4%)	11,744 (46.6%)
$KDDTest^{+}$	22,544	9711 (44%)	12,833 (56%)
$KDDTest^{-21}$	11,850	2152 (19.2%)	9698 (81.8%)

Table 2 Types of features present within NSL-KDD dataset

Feature number	Type	Feature name
1	Basic features	Duration
2		Protocol_type
3		Service
4		Flag
5		Src_bytes
6		Dst_bytes
7		Land
8		Wrong_fragment
9		Urgent
10		Hot
11	Content features	Num_failed _logins
12		Logged_in
13		Num_compromised
14		Root_shell
15		Su_attempted
16		Num_root
17		Num_file_creations
18		Num_shells
19		Num_access_files
20		Num_outbound_cmds
21		Is_hot_login
22		Is_guest_login
	Traffic features	
23	Same service features	Count
24		Srv_count
25		Serror_rate
26		Srv_serror_rate
27		Rerror_rate
28		Srv_rerror_rate
29		Same_srv_rate
30		Diff_srv_rate
31		Srv_diff_host_rate
32	Same host features	Dst_host_count
33		Dst_host_srv_count
34		Dst_host_same_srv_rate
35		Dst_host_diff_srv_rate
36		Dst_host_same_src_port_rate
37		Dst_host_srv_diff_host_rate
38		Dst_host_serror_rate
39		Dst_host_srv_serror_rate
40		Dst_host_rerror_rate
41		Dst_host_srv_rerror_rate

input to the encoder, which is encoded in the form of a function as shown in Eq. 1.

$$\mathrm{en}_{\mathrm{inp}} = func(d_{\mathrm{inp}}) \tag{1}$$

After encoding the input data, next phase is to decode the encoded results $(\mathrm{en}_{\mathrm{inp}})$ using Eq. 2.

$$\mathrm{dec}_{\mathrm{inp}} = func(\mathrm{en}_{\mathrm{inp}}) \tag{2}$$

Purpose of decoding is to reconstruct the original input. During the training phase, if auto-encoder only tries to reproduce its input to its output, then it may not be beneficial because it will lead to overfitting, and thus, results of

Table 3 Different types of attacks in NSL-KDD dataset

PROB	IP-sweep			Port-sweep	NMAP		Satan	
R2L	Warezmaster	Wareclient	Phf	SPY	Multi-HOP	Guess-password	FTP-Write	IMAP
U2R	Perl		Root-kit			Load-module	Butter-overflow	
DOS	Smurf	Land	Ping-0f-death		Neptune	Back	Teardrop	

unseen samples may be poor. To overcome the problem of overfitting during the training of individual auto-encoder, a sparsity term is introduced in the loss function which helps the auto-encoder to learn more generalized features during the pre-training phase.

A simple auto-encoder is comprised of an input layer, a hidden layer (for encoding the input data), and an output layer to reconstruct the original input. The loss function, used during pre-training of sparse auto-encoder, is described below.

$$\text{Loss function} = \frac{1}{S}\sum_{i=1}^{I}\sum_{j=1}^{J}\left(d_{\text{inp}_{ij}} - \text{dec}_{\text{inp}_{ij}}\right)^2 + \lambda_L \Omega_{\text{weight}} + \beta_L \Omega_{\text{sparsity}} \tag{3}$$

$\left(\frac{1}{S}\sum_{i=1}^{I}\sum_{j=1}^{J}\left(d_{\text{inp}_{ij}} - \text{dec}_{\text{inp}_{ij}}\right)^2\right)$ is the mean squared error between decoded $\left(\text{dec}_{\text{inp}}\right)$ and actual $\left(d_{\text{inp}}\right)$ inputs. Ω_{weight} is weight regularization with a coefficient λ_L, and Ω_{sparsity} is a sparsity regularization term with a coefficient β_L.L_2 weight regularization is expressed mathematically as:

$$\Omega_{\text{weight}} = \frac{1}{2}\sum_{l}^{L}\sum_{i}^{I}\sum_{j}^{J}\left(W_{ij}^{l}\right)^{2}. \tag{4}$$

In Eq. 4, L, I, and J show the number of hidden layers, instances, and variables, respectively, whereas W denotes weights of a link. L_2 weight regularization term in the loss function helps to control the weights of the network and thus act as regularization term (learns optimal weights during training phase). The sparsity regularization term is expressed mathematically as:

$$\begin{aligned}\Omega_{\text{sparsity}} &= \sum_{i'=1}^{s'} KL_{D'}\left(\frac{A'}{A'_{i'}}\right) \\ &= \sum_{i'=1}^{s'} A' \log\left(\frac{A'}{A'_{i'}}\right) + (1-A')\log\left(\frac{1-A'}{1-A'_{i'}}\right)\end{aligned} \tag{5}$$

$A_i^{'}$ shows the average activation value of a neuron i, whereas $A^{'}$ depicts the desired activation value of ith neuron. The activation value of a particular neuron is controlled by regulating the weights of the network. If the desired and the average activation values of a neuron are the same, then the sparsity regularization term will be zero. The sparsity regularization term increases with an increase in the difference between $A_i^{'}$ and $A^{'}$ values.

To make the proposed network deep, the decoders are removed after the individual training of auto-encoder and the encoded input is provided as input to the next auto-encoder. When the required number of auto-encoders is trained, the next step is to stack the individually trained auto-encoders to form a feedforward network (with good initial weights). This feedforward network is then fine-tuned using backpropagation-based learning algorithm. In the proposed DST-TL technique, features are extracted by implementing only unsupervised pre-training phase (using unlabeled data). Sparsity within deep sparse auto-encoder helps to extract effective feature space and improves generalization performance of the trained network. Moreover, features extracted from the proposed DST-TL-based approach, when combined with the original features of NSL-KDD dataset, generally lead to a diverse feature space.

3.1 Deep neural network and adaptive self-taught-based transfer learning (DST-TL) for IDS

The basic idea behind the proposed technique is the use of an unsupervised feature extraction using adaptive self-taught learning for developing an efficient IDS. Results depict that the proposed DST-TL technique provides good generalization in terms of different performance measures. Deep sparse auto-encoder is used in the proposed DST-TL technique for extracting effective features through unsupervised learning. Figure 1 shows a block diagram of the proposed DST-TL methodology.

3.2 Importance of transfer learning in the proposed DST-TL technique

Leveraging the knowledge gained from one type of machine learning problem and applying it to another machine learning task is known as transfer learning. The

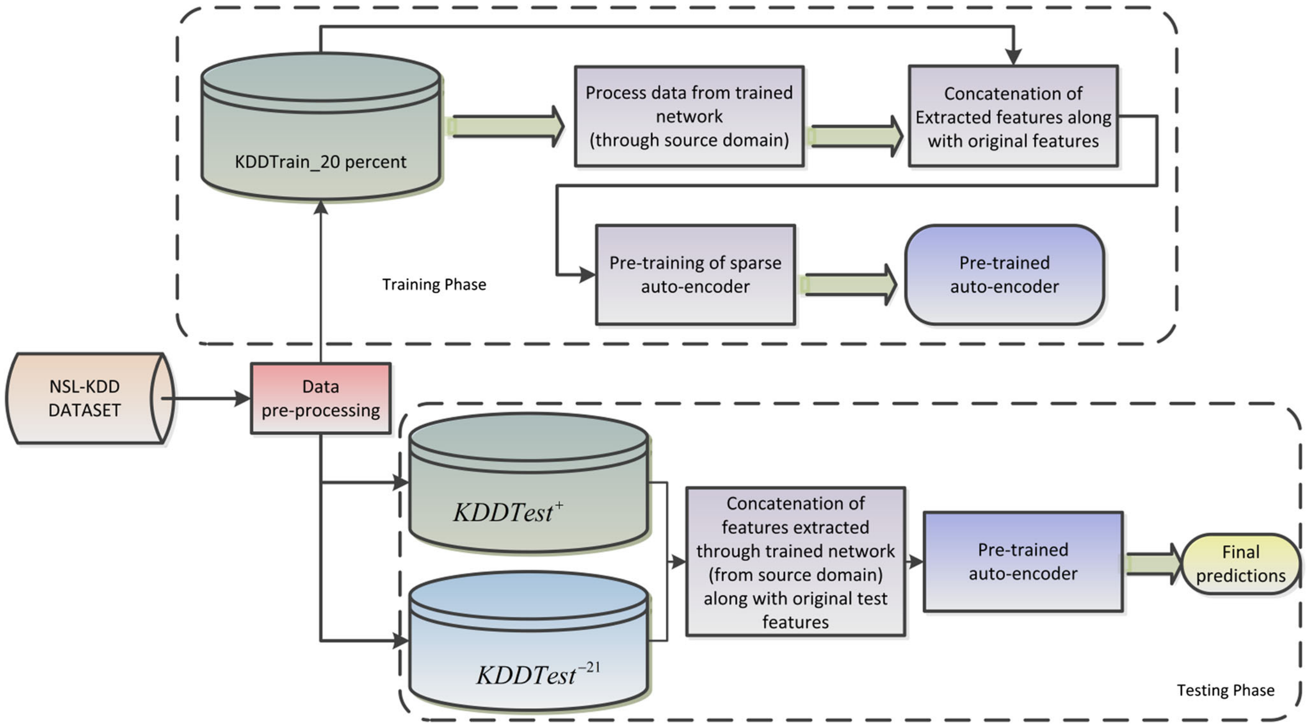
KDDTrain_20 percent
Process data from trained network (through source domain)
Concatenation of Extracted features along with original features
Pre-training of sparse auto-encoder
Pre-trained auto-encoder
Training Phase
NSL-KDD DATASET
Data pre-processing
KDDTest+
KDDTest-21
Concatenation of features extracted through trained network (from source domain) along with original test features
Pre-trained auto-encoder
Final predictions
Testing Phase

Fig. 1 Block diagram of the proposed DST-TL methodology

domain from where the knowledge is extracted is called the source domain. Extracted information from the source domain is either in the form of features or in the form of learning weights. Domain, in which the knowledge gained from the source domain is applied, is known as the target domain. Transfer learning can help in providing good results, particularly in case of deep neural networks where a lot of calibrated effort is required during training. There are several reasons that urge the machine learning experts to apply transfer learning. Some of the reasons are listed below:

- Sometimes labeled data are not sufficient to train a network in the target domain.
- A lot of effort (for tuning parameters) is required in the target domain to achieve remarkable results.
- Training from scratch takes a lot of time.

In case of transfer learning, it is not always a necessary condition that target and source domains follow the same distribution. There are also some forms of transfer learning approaches in which the target and source tasks are not from the similar domain. Some of the transfer learning approaches are given below:

- Self-taught learning
- Multitasking
- Domain adaptation
- Unsupervised transfer learning

In past, various transfer learning-based approaches [31–42] have been reported by many different researchers for machine learning-related tasks. In the proposed DST-TL technique, self-taught learning-based robust IDS is developed.

3.2.1 Self-taught learning

Typically, in transfer learning, target and source domains follow the same data distribution, and in comparison with the target domain, a sufficient amount of data is available in the source domain. In 2007, Rania et al. [31] presented the idea of the self-taught learning, in which unlabeled data in the source domain help in improving the performance of classification-related task in the target domain. Moreover, target and source domain data may not follow the same distribution. Features thus extracted using unlabeled data from a source domain may assist to improve learning in the target domain task.

3.2.2 Exploiting adaptive self-taught learning using deep sparse auto-encoder

Proposed DST-TL technique is comprised of two phases. In phase 1, a pre-trained deep sparse auto-encoder (using unlabeled data) on regression-related task (wind power prediction in this case) is used as a transferable knowledge from source domain. Only first and last layers (using pre-training in target domain) are added to the already trained hidden layers (from source domain) to form a feedforward neural network. Feature extraction in phase 1 is performed by just passing the target domain data through trained feedforward network.

In phase 2, original features along with extracted features (from phase 1) are provided as an input to train the sparse auto-encoder. Sparse auto-encoder is trained (using pre-training) such that only those features are extracted from the input data on which softmax classifier shows good performance (on validation data). Performance of trained network is then evaluated on $KDDTest^{+}$ and $KDDTest^{-21}$ test data. Figure 2 shows self-taught learning-based feature extraction methodology that is used in the proposed DST-TL technique.

3.2.3 Knowledge transfer from source to target domains

In self-taught learning, although the source task does not follow the same data distribution as that of the target domain, both domains are somehow related to each other. In the proposed DST-TL technique, source domain data are collected from wind farms located near Europe, and the task was to perform power forecasting against the provided features. As power forecasting at time t depends on the climatic condition of the previous hours; that is why, the predicted power and associated features of last 24 h are considered as input data to predict the power of time t. In short, in the source domain, the dataset is of the time series nature. Moreover, wind power prediction is a challenging task because any sudden fluctuation in the geographical and climatic conditions can affect the generated power. On the other hand, the target domain task in the proposed DST-TL technique is related to intrusion detection, in which NSL-KDD is used as benchmark dataset. Discussion related to the NSL-KDD dataset in Sect. 2 shows that traffic features are also of the time series nature and are calculated on the basis of connections in the past 2 s. When an intruder attacks the network, it results in the sudden change in the behavior of the Network flow, so any sudden change in the behavior of the network may be the sign of intrusion within the network. Network intrusion detection, just like wind power, is of unpredictable nature. Thus, the source domain task is related to target domain task in terms of the time series nature of input features and unpredictable behavior of power and intrusion.

In the proposed DST-TL technique, because of the commonality between the target and source domains, trained network from source domain is efficient enough to predict the wind power despite any abrupt change in the atmospheric behavior and thus share the gained knowledge to target domain.

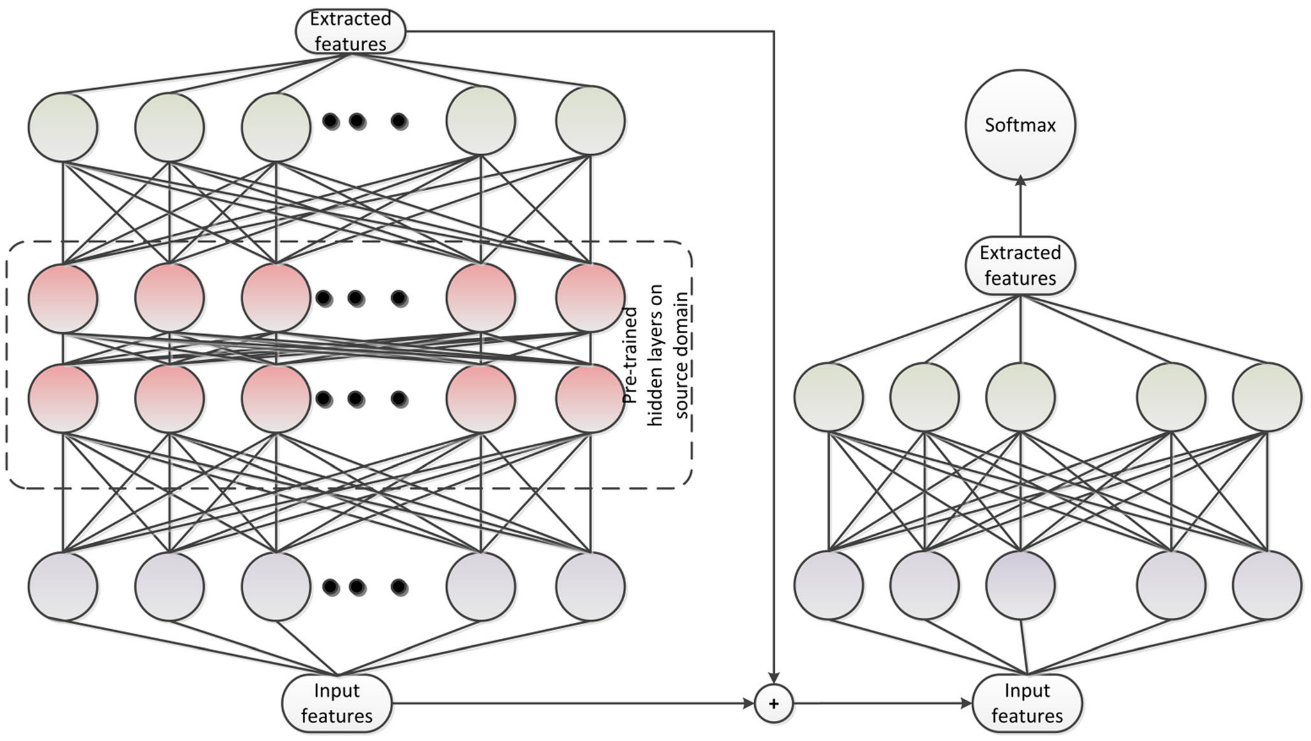

Fig. 2 Use of self-taught learning in training of deep sparse auto-encoder

4 Implementation details

All the experimental work related to the proposed work was performed on a desktop computer, having 16 GB RAM, Intel(R) Core(TM) i7-33770, 64-bit operating system, CPU@3.4 GHz. The operating system used was Windows 7 professional, and MATLAB 2016 a is used as a programming tool.

4.1 Parameter setting of the proposed technique

In order to set the parameters of deep sparse auto-encoder, 10% of training data were used for validating the optimal parameters. Table 4 shows the parameter setting of the deep sparse auto-encoder, in which hidden layers are pre-trained on source domain task. Table 5 illustrates the parameter setting during the training phase of sparse auto-encoder with (using original as well as extracted features from the source domain) or without (using only original features of NSL-KDD data) self-taught learning approach.

4.2 Performance evaluation

To evaluate the proposed DST-TL technique, detection rate, false alarm rate, area under the ROC curve (AUC-ROC), area under the precision–recall curve (AUC-PR), and accuracy are used as evaluation measures. AUC-ROC is basically the plot of sensitivity and specificity, at different values of the thresholds. However, AUC-PR is the plot between precision and recall. Mathematically, the measures are defined in Eqs. 6–11.

$$\text{Sensitivity} = \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6}$$

$$1 - \text{Sepecificity} = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{7}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8}$$

$$\text{Detection Rate} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{9}$$

$$\text{False Alarm Rate} = \frac{\text{FP}}{\text{TN} + \text{FP}} \tag{10}$$

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \times 100 \tag{11}$$

In the above equations, TP and TN are the number of positive and negative class samples, respectively, that are correctly classified by the classifier. FP and FN are the number of negative and positive class samples, respectively, which are classified wrongly by the classifier.

5 Results and discussion

In the proposed DST-TL technique, after the training of a sparse auto-encoder, the performance of the trained model is evaluated on $KDDTest^{+}$ and $KDDTest^{-21}$ datasets. To show the stability of the proposed DST-TL technique, performance in terms of detection and false alarm rates, AUC-ROC and AUC-PR, and accuracy for ten independent runs is shown in Tables 8, 9, and 10, respectively. Before checking the performance of proposed DST-TL technique, the performance of the conventional classifiers [multilayer perceptron (MLP), nonlinear principal component analysis (NLPCA), and deep belief network (DBN)] is evaluated on $KDDTest^{+}$ and $KDDTest^{-21}$ datasets. MLP is a commonly used classifier that uses backpropagation learning algorithm during training. DBN is a type of classifier which is formed by stacking of independently trained RBM (restricted Boltzmann machine). After stacking of RBM, the backpropagation learning algorithm is used to fine tune the stacked feedforward network. Principal component analysis (PCA) is a commonly used feature reduction technique; however, NLPCA is considered as a generalized form of PCA. In this case, NLPCA comprises of a simple auto-encoder having an encoding layer that contains less neurons in comparison with the number of neurons in input layer. Features that are extracted from the hidden layer are provided as input to a simple classifier for classifying the

Table 4 Parameter setting of deep sparse auto-encoder (for extracting features) using hidden layers trained on source domain task

Parameters	Layer trained according to the target domain	Trained layers on regression-related task in the source domain		Layer trained according to the target domain
	Layer 1	Layer 2	Layer 3	Layer 4
Coefficient of L_2 weight regularization	0.0002	0.00003	0.00001	0.00001
Coefficient of sparsity regularization	4	4	4	4
Maximum number of epochs	200	500	250	150
Number of neurons	124	300	220	200

Table 5 Parameter setting of the sparse auto-encoder trained on the combined original and extracted features

Parameters	Without self-taught learning approach		With self-taught learning approach	
	Layer 1	Layer 2	Layer 1	Layer 2
Coefficient of L_2 weight regularization	0.0002	0.00002	0.00002	0.00002
Coefficient of sparsity regularization	5	4	4	3
Maximum number of epochs	100	200	200	100
Number of neurons	100	80	166	180

Table 6 Performance comparison of MLP, NLPCA, and DBN in terms of AUC-ROC, AUC-PR, and detection and false alarm rates

	AUC-ROC		AUC-PR		Detection rate		False alarm rate	
	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$	$KDDTest^{+}$	$KDDTest^{-21}$
MLP	0.88 ± 0.02	0.75 ± 0.02	0.92 ± 0.01	0.93 ± 0.01	0.65 ± 0.01	0.28 ± 0.01	0.38 ± 0.02	0.51 ± 0.03
DBN	0.85 ± 0.03	0.74 ± 0.07	0.90 ± 0.02	0.92 ± 0.02	0.65 ± 0.01	0.27 ± 0.01	0.40 ± 0.01	0.53 ± 0.02
NLPCA	0.89 ± 0.03	0.64 ± 0.03	0.92 ± 0.01	0.91 ± 0.01	0.66 ± 0.01	0.25 ± 0.01	0.35 ± 0.01	0.47 ± 0.02

input data. Table 6 shows the performance of MLP, DBN, and NLPCA in terms of all the evaluation measures. Parameters that are used during the training are provided in Table 7.

5.1 Performance evaluation using detection and false alarm rates

Table 8 shows the comparison between detection and false alarm rates of a sparse auto-encoder with and without self-taught learning. The standard deviation of error for the ten independent runs is also shown. Table 8 illustrates that deep, sparse auto-encoder trained with self-taught learning approach shows better and more stable results (evident from the low values of standard deviation of error) in comparison with the deep sparse auto-encoder trained without self-taught learning.

5.2 Performance evaluation using AUC-ROC, AUC-PR, and accuracy

Table 9 shows the performance of the proposed DST-TL on $KDDTest^{+}$ and $KDDTest^{-21}$ datasets in terms of AUC-ROC and AUC-PR. It can be observed that the self-taught learning approach increases stability and performance of network on unseen data and also has a low standard

Table 7 Parameter setting of MLP, NLPCA, and DBN during training

MLP	DBN							NLPCA
Number of neurons in the hidden layer	No of neurons			No of epochs	Batch size	Momentum	Learning rate	No of neurons in encoding layer
	Layer 1	Layer 2	Layer 3					
25	80	75	50	50	2	0.04	0.001	18

Table 8 Performance comparison of the trained sparse auto-encoders for ten independent runs

	Spars auto-encoder without transfer learning		Sparse auto-encoder with transfer learning	
	Detection rate	False alarm rate	Detection rate	False alarm rate
$KDDTest^{+}$	0.86 ± 0.09	0.13 ± 0.09	0.85 ± 0.05	0.14 ± 0.05
$KDDTest^{-21}$	0.81 ± 0.13	0.18 ± 0.13	0.80 ± 0.06	0.19 ± 0.06

Table 9 Performance comparison of trained sparse auto-encoders on test data

	Sparse deep auto-encoder without transfer learning		Deep sparse auto-encoder with transfer learning	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
KDDTest^{+}	0.89 ± 0.01	0.91 ± 0.01	0.91 ± 0.01	0.93 ± 0.01
KDDTest^{-21}	0.61 ± 0.03	0.88 ± 0.01	0.69 ± 0.01	0.91 ± 0.01

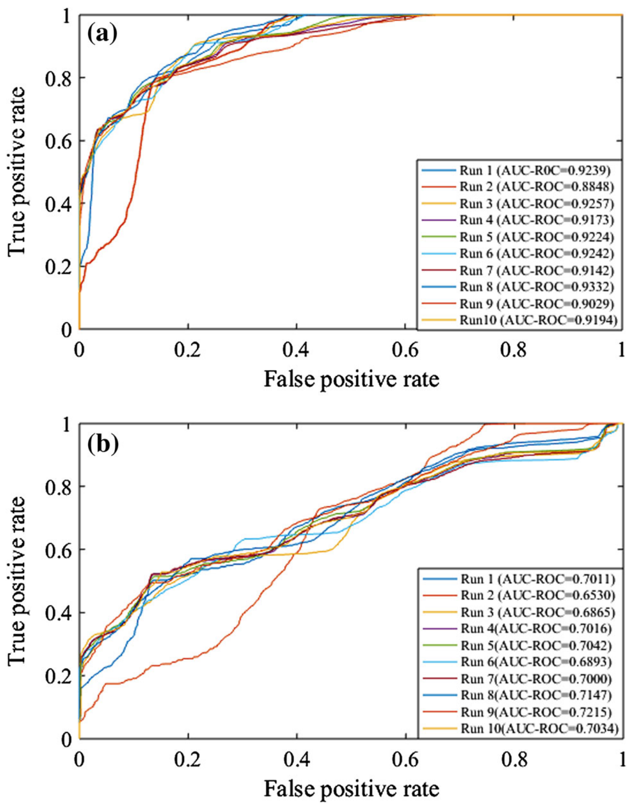

Fig. 3 AUC-ROC of the proposed DST-TL technique for ten independent runs, **a** on *KDDTest*$^{+}$ and **b** on *KDDTest*$^{-21}$

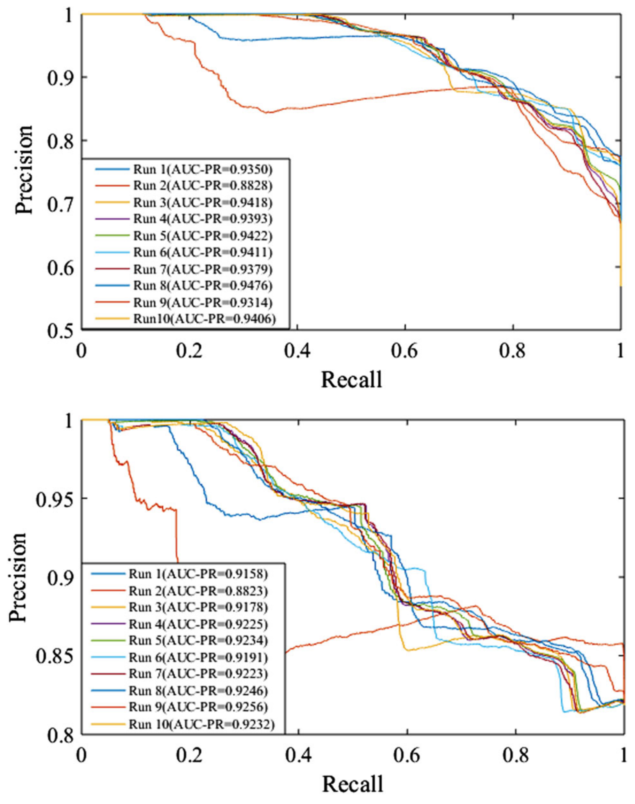

Fig. 4 AUC-PR of ten independent runs, **a** on *KDDTest*$^{+}$ and **b** on *KDDTest*$^{-21}$

deviation of error. Figures 3 and 4 graphically illustrate the AUC-ROC and AUC-PR on *KDDTest*$^{+}$ and *KDDTest*$^{-21}$ for ten independent runs.

Table 10 shows the accuracy of the proposed technique for ten independent runs. It is observed that features extracted through self-taught learning approach lead to the better training of the proposed DST-TL method. The self-taught learning approach helps in extracting robust features, which are then utilized by the deep sparse auto-encoder to learn optimal weights during training and thus increases the generalization performance.

5.3 Performance comparison of the proposed DST-TL technique with state-of-the-art techniques

Table 11 shows a comparison of the proposed DST-TL method with different classifiers (J48, naïve Bayes, NB tree, random forest, random tree, MLP, NLPCA, DBN, and SVM) and also with fuzzy-based semi-supervised technique. Table 11 depicts that the performance of the proposed DST-TL technique is better in comparison with existing methods, especially on *KDDTest*$^{-21}$ dataset.

Table 10 Performance comparison for ten independent runs of the trained sparse auto-encoders in terms of accuracy

	Sparse auto-encoder without transfer learning (accuracy)		Sparse auto-encoder trained using transfer learning (accuracy)	
	KDDTest^{+}	*KDDTest*$^{-21}$	*KDDTest*$^{+}$	*KDDTest*$^{-21}$
1	**85.30**	**74.50**	**84.60**	**79.90**
2	84.50	72.70	81.70	71.40
3	80.60	63.30	82.40	69.40
4	74.30	78.30	82.40	72.40
5	80.60	63.40	82.60	71.50
6	75.80	75.50	83.80	74.90
7	72.50	80.70	82.20	72.10
8	76.50	73.90	83.80	72.30
9	77.40	57.40	81.80	69.40
10	73.30	79.20	82.40	77.70
	78.08 ± 4.27	**71.89 ± 7.43**	**82.77 ± 0.91**	**73.1 ± 3.09**

Bold values show the best and average results (that depict how stable is the proposed methodology) that are obtained for ten independent runs

Table 11 Performance comparison of the proposed technique with the existing methods

Classifier	Accuracy	
	KDDTest^{+}	*KDDTest*$^{-21}$
J48	81.05	63.97
Naïve Bayes	76.56	55.77
NB tree	82.02	66.16
Random forest	80.67	63.25
Random tree	81.59	58.51
MLP	77.41	57.34
NLPCA	76.73	56.58
DBN	76	54.55
SVM	69.52	42.29
Fuzziness-based IDS (experiment 2)	84.12	68.82
DST-TL (best accuracy in ten independent runs)	**84.60**	**79.90**
DST-TL (average performance)	**82.77 ± 0.91**	**73.1 ± 3.09**

Bold values show the best and average results (that depict how stable is the proposed methodology) that are obtained for ten independent runs

6 Conclusion

A novel network IDS based on deep sparse auto-encoder that exploits self-taught learning is proposed. In the first phase, feature extraction is performed by passing the original feature set of NSL-KDD through the pre-trained network. Then, a combination of original and extracted features is used to train the sparse auto-encoder. The combined features improve the effectiveness of the feature space and thus increase the performance of the sparse auto-encoder on test samples. It is experimentally shown that the proposed DST-TL technique yields improved performance, although adaptation of the network trained on regression-related task is used to improve the performance of the IDS. The experimental comparison shows that the sparse auto-encoder trained on the improved feature space extracted through self-taught learning is more robust and stable in comparison with the sparse auto-encoder trained on the original feature space. Performance on test data shows that in comparison with previous techniques, the proposed DST-TL approach is robust and provides improved prediction accuracy. In future, we intend to apply deep neural networks, especially recent architectures of deep convolutional neural networks for classifying different types of attacks [43].

Acknowledgements This work is supported by the Higher Education Commission of Pakistan under the Indigenous Ph.D. Scholarship (PIN#213-54573-2EG2-097) Program. We also acknowledge Pakistan Institute of Engineering and Applied Sciences (PIEAS) for healthy research environment which led to the research work presented in this article.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Ashoor AS, Gore S (2011) Importance of intrusion detection system (ids). Int J Sci Eng Res 2:1–4
2. Scarfone K, Mell P (2007) Guide to intrusion detection and prevention systems (idps). NIST Spec Publ 800:94
3. Letou K, Devi D (2015) Host-based intrusion detection and prevention system (HIDPS). Int J Comput Appl 69:975–988. https://doi.org/10.5120/12136-8419
4. Mukherjee B, Heberlein LT (1994) Network intrusion detection. IEEE Netw 83:26–41
5. Masri W, Podgurski A (2008) Application-based anomaly intrusion detection with dynamic information flow analysis 5. Comput Secur 27:176–187. https://doi.org/10.1016/j.cose.2008.06.002
6. Dı J (2009) Anomaly-based network intrusion detection: techniques, systems and challenges. Comput Secur 28:18–28. https://doi.org/10.1016/j.cose.2008.08.003
7. Diaz-Gomez PA, Hougen DF (2007) Misuse detection: an iterative process vs. a genetic algorithm approach. In: ICEIS (2), pp 455–458
8. Malhotra S, Bali V, Paliwal KK (2017) Genetic programming and K-nearest neighbour classifier based intrusion detection model. In: 7th International conference on cloud computing data science and engineering 2017, IEEE, pp 42–46
9. Zhang C, Jiang J, Kamel M (2005) Intrusion detection using hierarchical neural networks. Pattern Recognit Lett 26:779–791. https://doi.org/10.1016/j.patrec.2004.09.045
10. Panda M, Patra MR (2007) Network intrusion detection using naive bayes. Int J Comput Sci Netw Secur 7:258–263
11. Zhang J, Zulkernine M (2005) Network intrusion detection using random forests. PST, Citeseer
12. Portnoy L, Eskin E, Stolfo S (2001) Intrusion detection with unlabeled data using clustering. In: Proceedings of the ACM CSS workshop on data mining applied to security (DMSA-2001, Citeseer)
13. Leung K, Leckie C (2005) Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of twenty-eighth Australasian computer science conference, vol 38, pp 333–342
14. Chizari BMARRM, Eslami AMM (2016) A hybrid method consisting of GA and SVM for intrusion detection system. Neural Comput Appl 27:1669–1676. https://doi.org/10.1007/s00521-015-1964-2
15. Devaraju SRS (2017) Attack' s feature selection-based network intrusion detection system using fuzzy control language. Int J Fuzzy Syst 19:316–328. https://doi.org/10.1007/s40815-016-0160-6
16. Srinivasan T, Vijaykumar V, Chandrasekar R (2006) A self-organized agent-based architecture for power-aware intrusion detection in wireless ad-hoc networks. In: 2006 International conference on computing and informatics, 2006, pp 1–6. https://doi.org/10.1109/icoci.2006.5276609
17. Puri A, Sharma N (2017) A novel technique for intrusion detection system for network security using hybrid svm-cart. Int J Eng Dev Res 5:155–161
18. Mukkamala S, Janoski G, Sung A (2002) Intrusion detection using neural networks and support vector machines. In: Proceedings of the 2002 international joint conference on neural networks, IJCNN'02, 2002, pp 1702–1707
19. Aamir R, Ashfaq R, Wang X, Zhexue J, Abbas H, He Y (2017) Fuzziness based semi-supervised learning approach for intrusion detection system. Inf Sci 378:484–497. https://doi.org/10.1016/j.ins.2016.04.019
20. Kim J, Shin N, Jo SY, Kim SH. Method of intrusion detection using deep neural network. In: 2017 IEEE International conference on big data and smart computing (BigComp), IEEE; 2017, pp 313–316
21. Kevric J, Jukic S, Subasi A (2016) An effective combining classifier approach using tree algorithms for network intrusion detection. Neural Comput Appl. https://doi.org/10.1007/s00521-016-2418-1
22. Cup KDD (2007) Data: available on https://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data. Accessed 25 Mar 2019
23. Tavallaee M, Bagheri E, Lu W, Ghorbani AA (2009) A detailed analysis of the KDD CUP 99 data set. In: Computational intelligence in security and defence application, 2009, pp 1–6
24. Lu C, Min H, Gui J, Zhu L, Lei Y (2013) Face recognition via weighted sparse representation. J Vis Commun Image Represent 24:111–116. https://doi.org/10.1016/j.jvcir.2012.05.003
25. Mi J, Lei D, Gui J (2013) Optik a novel method for recognizing face with partial occlusion via sparse representation. Opt Int J Light Electron Opt 124:6786–6789. https://doi.org/10.1016/j.ijleo.2013.05.099
26. Gui J, Liu T, Tao D, Sun Z, Tan T (2016) Representative vector machines: a unified framework for classical classifiers. IEEE Trans Cybern 46:1877–1888
27. Gui J, Tao D, Sun Z, Luo Y, You X, Tang YY (2014) Group sparse multiview patch alignment framework with view consistency for image classification. IEEE Trans Image Process 23:3126–3137
28. Gui J, Sun Z, Hou G, Tan T. An optimal set of code words and correntropy for rotated least squares regression. In: IEEE international joint conference on biometrics (IJCB), 2014, pp 1–6
29. Qureshi AS, Khan A, Zameer A, Usman A (2017) Wind power prediction using deep neural network based meta regression and transfer learning. Appl Soft Comput J 58:742–755. https://doi.org/10.1016/j.asoc.2017.05.031
30. Gui J, Sun Z, Ji S, Member S, Tao D, Tan T (2017) Feature selection based on structured sparsity: a comprehensive study. IEEE Trans Neural Netw Learn Syst 28:1490–1507
31. Raina R, Battle A, Lee H, Packer B, Ng AY (2007) Self-taught learning: transfer learning from unlabeled data. In: Proceedings of the 24th international conference on machine learning, ACM, 2007, pp 759–66
32. Maurer A, Pontil M, Romera-Paredes B. Sparse coding for multitask and transfer learning. In: International conference on machine learning, 2013, pp 343–351
33. Lim JJ, Salakhutdinov RR, Torralba A (2011) Transfer learning by borrowing examples for multiclass object detection. In: Advances in neural information processing systems, 2011, pp 118–126
34. Hu Q, Zhang R, Zhou Y (2016) Transfer learning for short-term wind speed prediction with deep neural networks. Renew Energy 85:83–95. https://doi.org/10.1016/j.renene.2015.06.034
35. Du B, Zhang L, Tao D, Zhang D (2013) Neurocomputing unsupervised transfer learning for target detection from hyperspectral images. Neurocomput J 120:72–82. https://doi.org/10.1016/j.neucom.2012.08.056
36. Cao X (2013) A practical transfer learning algorithm for face verification. In: IEEE international conference on computer vision (ICCV), 2013, pp 3208–3215. https://doi.org/10.1109/iccv.2013.398
37. Yang S, Lin M, Hou C (2012) A general framework for transfer sparse subspace learning. Neural Comput Appl. https://doi.org/10.1007/s00521-012-1084-1

38. La L, Guo Q, Cao Q, Wang Y (2014) Transfer learning with reasonable boosting strategy. Neural Comput Appl 24:807–816. https://doi.org/10.1007/s00521-012-1297-3
39. Yang S, Hou C, Zhang C (2013) Robust non-negative matrix factorization via joint sparse and graph regularization for transfer learning. Neural Comput Appl. https://doi.org/10.1007/s00521-013-1371-5
40. Seera M, Peng C (2014) Transfer learning using the online fuzzy min–max neural network. Neural Comput Appl. https://doi.org/10.1007/s00521-013-1517-5
41. Silva M, Cardoso JS (2017) Multi-source deep transfer learning for cross-sensor biometrics. Neural Comput Appl. https://doi.org/10.1007/s00521-016-2325-5
42. Khan A, Sohail A, Ali A (2018) A new channel boosted convolution neural network using transfer learning. arXiv:180408528
43. Khan A et al (2019) A survey of the recent architectures of deep convolutional neural networks. arXiv:1901.06032