Enhanced Network Anomaly Detection Based on Deep Neural Networks

Sheraz Naseer, Yasir Saleem, Shehzad Khalid, M Khawar Bashir, Jihun Han, M Munwar Iqbal and Kijun Han

***Abstract*—Due to the monumental growth of Internet applications in last decade, the need for security of information network has increased manifolds. As a primary defense of network infrastructure, an Intrusion Detection System is expected to adapt to dynamically changing threat landscape. Many supervised and unsupervised techniques have been devised by researchers from the discipline of machine learning and data mining to achieve reliable detection of anomalies. Deep learning is an area of Machine Learning which applies neuron-like structure for learning tasks. Deep learning has profoundly changed the way we approach learning tasks by delivering monumental progress in different disciplines like Speech Processing, Computer Vision and Natural Language Processing to name a few. It is only relevant that this new technology must be investigated for information security applications. The aim of this article is to investigate the suitability of Deep learning approaches for anomaly-based intrusion detection system. For this study, we developed anomaly detection models based on different deep neural network structures including Convolutional Neural Networks, Autoencoders and Recurrent Neural Networks. These deep models were trained on NSLKDD training dataset and evaluated on both test datasets provided by NSLKDD namely NSLKDDTest+ and NSLKDDTest21. All experiments in this study are performed by authors on a GPU-based test bed. Conventional Machine Learning based intrusion detection models were implemented using well-known classification techniques including Extreme Learning Machine, Nearest Neighbor, Decision-Tree, Random-Forest, Support Vector Machine, Naive-Bays, and Quadratic Discriminant Analysis. Both deep and conventional Machine Learning models were evaluated using well-known classification metrics including, Receiver Operating Characteristics, Area under Curve, Precision-Recall Curve, mean average precision and accuracy of classification. Experimental results of deep IDS models showed promising results for real-world application in anomaly detection systems.**

***Index Terms*—Deep Learning, Convolutional Neural Networks, Autoencoders, LSTM, k_NN, Decision_Tree, Intrusion Detection, Convnets, Information Security**

I. Introduction

Sheraz Naseer is a graduate student at Department of CS&E, University of Engineering and Technology (UET), Lahore, Pakistan and working at University of Management and Technology (UMT), Lahore, Pakistan.

Dr. Yasir Saleem is with Department of CS&E, University of Engineering and Technology (UET), Lahore, Pakistan.

Shehzad Khalid is with Department of Computer Engineering, Bahria University, Islamabad, Pakistan.

M Khawar Bashir is a graduate student at Department of CS&E, University of Engineering and Technology (UET), Lahore, Pakistan and working at University of Veterinary and Animal Sciences, Lahore, Pakistan

Jihun Han and Kijun Han are with School of Computer Science and Engineering, Kyungpook National University, Daegu, S. Korea. Kijun Han is the corresponding Author. **Corresponding email: kjhan@knu.ac.kr**

M Munwar Iqbal is with Department of CS, University of Engineering and Technology (UET), Taxila, Pakistan.

Manuscript received May 02, 2018; revised June 26, 2018.

NETWORK intrusion detection refers to the problem of monitoring and differentiating such network flows and activities from the normal expected behavior of network which can adversely impact the security of information systems. The search of reliable solutions by Governments and organizations to protect their information assets from unauthorized disclosures and illegal accesses has brought intrusion detection and prevention at the forefront of information security landscape.
Denning [1] proposed the idea of developing intrusion detection system by employing Artificial Intelligence techniques on security events to identify abnormal usage patterns and intrusions. This idea pioneered a new breed of intrusion detection systems which were based on learning algorithms rather than always-updating signatures of intrusions. Over the last three decades, machine learning techniques were applied as a conventional approach for developing network anomaly detection models. These approaches employ supervised, unsupervised and semi-supervised learning algorithms to propose solutions for anomaly detection problem.
Anomaly detection is modeled as a classification problem in supervised learning. Supervised learning uses labeled data to train anomaly detection models. The goal of this type of training is to classify the test data as anomalous or normal on the basis of feature vectors. Unsupervised learning, on the other hand, uses unlabeled or untagged data to perform the task learning. One of the popular unsupervised learning technique is clustering [2], which searches for similarities among instances of the dataset to build clusters. Instances sharing related characteristics are assumed to be alike and placed in the same cluster. Semi-Supervised Learning (SSL) is a combination of supervised and unsupervised learning. The SSL approach utilizes both labeled and unlabeled data [3] for learning. SSL methods learn feature-label associations from labeled data and assign the labels to unlabeled instances having similar features that of a labeled instance on the basis of learned feature-label associations.

Deep Learning is an area of Machine Learning which applies neuron like mathematical structures [4] for learning tasks. Neural Networks have been around for many decades [5] and have been gaining and losing the favor of research community. The latest rise of this technology is attributed to Alexnet [6], a Deep Neural Network, which won the ImageNet classification challenge. Alexnet achieved top-1 and top-5 error rates of 37.5 % and 17.0% on ImageNet Dataset [7] which were considerably better than the previous state-of-the-art mechanisms. Since then, Deep Neural Networks (DNNs) have attracted the attention of research community once again and multiple DNN structures including Convolutional neural networks (CNNs) [8], Recurrent Neural networks (LSTM) [9], Deep belief nets (DBNs) and different types of Autoencoders including Sparse , Denoising [10], Convolutional[11], Contractive [12] and variational Autoencoders have been proposed. These DNN structures have been successfully applied to devise state of the art solutions in multiple disciplines.

Application of Deep Neural Networks for the solution of Information security problems is a relatively new area of research. We observed three research gaps during literature review of anomaly detection problem. The first gap was lack of investigation of well-known deep learning approaches for anomaly detection. Although isolated studies were available as described in II, no comprehensive research work was available to fill this gap. The second research gap was the use of training datasets for both training and testing of models using cross-validation mechanisms. Most of the recent works followed this approach and reported very high detection rates, e.g., Kim et al. [15] used a four-layer DNN with 100 units for intrusion detection on the KDD99 dataset and reported 99% accuracy. We believe that this approach does not provide a reliable solution of anomaly detection problem, as, given sufficient training, models can be over-fitted to achieve such high rates. The 3rd gap turned out to be lack of comparison/evaluation of deep learning models amongst themselves and with conventional machine learning based models using standardized classification quality metrics which was a natural consequence of previous two gaps. Standardized classification quality metrics include RoC Curve, Area under RoC, Precision-Recall Curve, mean average precision (mAP) and accuracy of classification.

The primary contribution of this work is filling abovementioned research gaps by designing and implementing anomaly detection models based on state of the art Deep Neural Networks and their evaluation using standardized classification quality metrics. The first gap is filled by developing anomaly detection models using Deep CNN, LSTM and multiple types of Autoencoders. To the best of our knowledge, the DNN structures (DCNN, Contractive, and Convolutional Autoencoders) investigated in this study have not been analyzed for anomaly detection. In addition, comparisons of deep learning based anomaly detection models are provided with well-known classification schemes including SVM, K-NN, Decision-Tree, Random-Forest, QDA and Extreme Learning machine. To fill second research gap, we opted to train all models on training dataset without ever exposing test dataset to the model during training and then tested/evaluated the models on testing datasets. This approach provided a fair estimate of model capabilities by using unseen data instances at evaluation time. To bridge the third research gap, Deep learning based anomaly detection models were evaluated amongst themselves and with conventional machine learning models by using unseen test data and employing standard classification quality metrics including RoC Curve, Area under RoC, Precision-Recall Curve, mean average precision (mAP) and accuracy of classification.

All experiments in this study are performed by authors on NSLKDD dataset provided by [16] using a GPU-powered test-bed. NSLKDD is derived from KDDCUP99 [17] which was generated in 1999 from the DARPA98 network traffic. Tavallaee et al. [16] discovered some inherent flaws in original KDDCUP99 dataset which had adverse impacts on the performance of IDS models trained and evaluated on the Dataset. A statistically enhanced version of dataset called NSLKDD was proposed by [16] to counter discovered statistical problems. Some advantages of NSLKDD over KDDCUP99 dataset include removal of redundant records from training dataset for reducing complexity and bias towards frequent records and the introduction of non-duplicate records in testing datasets for unbiased evaluation.

NSLKDD Dataset is available in four partitions. Two partitions namely NSLKDDTrain20p and NSLKDDtrain+ serve as training Dataset for model learning and provide 25,192 and 125,973 training records respectively. Remaining two partitions called NSLKDDTest+ and NSLKDDTest21 are available for performance evaluation of trained models on unseen data and provide 22,543 and 11,850 data instances respectively. Additionally, NSLKDDTest21 contains records for attack types not available in other NSLKDD train and test Datasets. These attack types include *processtable*, *mscan*, *snmpguess*, *snmpgetattack*, *saint*, *apache2*, *httptunnel*, *back* and *mailbomb*. All models in our study were trained on NSLKDD training datasets (NSLKDDTrain20p and NSLKDDTrain+) and tested on NSLKDD test datasets (NSLKDDTest+ and NSLKDDTest21). This approach was also adopted by [18], [19] and [20].

Like its predecessor, NSLKDD dataset consists of 41 input features as well as class labels. Features 1 to 9 represent the basic features which were created from TCP/IP connection without payload inspection. Features 10 to 22 comprised of content features, generated from the payload of TCP segments of packets. Features 23 to 31 were extracted from time-based traffic properties while features 32 to 41 contain application based traffic features that were designed to measure attack within intervals longer than 2 seconds. A class label was provided with each record, which identified the network traffic instance either as normal or an attack. Original KDDCUP99 dataset listed different types of attacks shown in Table I.

Rest of the article is divided into VI sections. Section II highlights prominent works related to IDS problem. Section III provides the architectural designs of DNNs used in this study. Section IV sheds light on implementation details including hardware setup and software tool-chain. In Section V, we present results of proposed DNN based models along with comparisons of results and timing information. This section is followed by section V and VI which describe the conclusion and references of research respectively.

TABLE I: Attack Types in KDDCUP99 Dataset

Denial of Service (DoS)	User to Root (U2R)	Remote to Local (R2L)	Probing (Probe)
Back	Buffer Overflow	FTP write	IPSweep
Land	Load module	Guess Password	NMAP
Neptune	Perl	IMAP	Port Sweep
Ping of Death	Rootkit	MultiHop	Satan
Smurf		Phf	
TearDrop		SPY	
		Warezclient	
		WarezMaster	

II. Related Works

Different Machine learning techniques including supervised, unsupervised and semi-supervised, have been proposed to enhance the performance of anomaly detection. Supervised approaches such as k-nearest neighbor (k-NN) [21], neural networks and support vector machine (SVM) [23] have been studied extensively for anomaly detection. Laskov et al. [24] provided a comparative analysis of supervised and unsupervised learning techniques with respect to their detection accuracy and ability to detect unknown attacks. Ghorbani et al. [25] provided a comprehensive review of supervised and unsupervised learning approaches for anomaly detection. Solanas et al. [26] presented clustering algorithms for anomaly detection. A comprehensive repertoire of anomaly-based intrusion detection systems is presented by Bhattacharraya et al. in [27] and Tavallee [18] compared the performance of the NSLKDD dataset on different classification algorithms including Naive-Bayes, Support Vector Machines, and Decision-Trees, etc.

Application of Deep Neural Networks for the solution of Information security problems is a relatively new area of research. DNN structures like Autoencoders (AE), Deep Belief Networks (DBNs) and LSTM have been used for Anomaly Detection. Gao et al. [30] proposed an IDS architecture based on DBNs using energy based reduced Boltzmann machines (RBMs) on KDDCup99 Dataset. Wang [31] proposed a deep network of stacked autoencoders (SAE) for network traffic identification. A semi-supervised learning based approach with Random weights based NN (NNRw) is used by Ashfaq et al. [19] to implement an IDS architecture using NSLKDD.

Aygun et al. [32] employed vanilla and denoising deep Autoencoders on NSLKDD and claimed accuracy of 88.28% and 88.6% on NSLKDDtest+ dataset. They did not provide results for NSLKDDtest21 dataset neither did they provide any other quality metrics of their classifier including AuROC, Precision, Recall, and mAP. Yousefi-Azar et al. [20] used Autoencoders as an unsupervised latent feature generation mechanism and provided a comparison of classifiers using conventional NSLKDD features and possible representations of NSLKDD. They reported the accuracy of 83.34% by using latent representations of NSLKDD. Alom et al. [33] trained a Deep Belief Network (DBN) using staked Restricted-Boltzmann Machines (RBMs) and reported the accuracy of 97.5% on NSLKDD training dataset (training accuracy). They did not provide results for either NSLKDDTest+ or NSLKDDTest21 datasets. Hodo et al. [34] provided a taxonomy and survey of deep and conventional structures for intrusion detection. Javaid et al. [35] developed a "Self-taught Learning" classification mechanism by combining encoder layers of Sparse Autoencoder (for hidden features extraction) with Softmax regression (for probability estimates of classes) to perform classification on NSLKDD. They reported 98.3% accuracy on training data (training accuracy) for two-class classification, 98.2% training accuracy for 5 class classification and 98.8% training accuracy for 23 class classification. Javaid et al. provided results for neither NSLKDDTest+ nor NSLKDDtest21 datasets. Bontemps et al. [36] proposed an LSTM based anomaly detection system and reported variations incorrect and false alarms of prediction errors with a change of a β parameter of the proposed system but did not provide results in well-known metrics. This shows the need for an experimental study which develops anomaly detection models using well-known Deep learning approaches and evaluates them on previously unseen test datasets using standardized classification quality metrics. In this study, we aim to close above-mentioned research gap and evaluate Deep learning based models on unseen data using standard classification quality metrics including RoC Curve, Area under RoC, Precision-Recall Curve, mean average precision (mAP) and accuracy of classification.

III. Preliminaries

In this section, a brief overview of DNN structures, implemented in this article, is provided. The DNNs discussed in this section include:

1) Different Autoencoders (Sparse, Denoising, Contractive, Convolutional)
2) LSTM
3) Convolutional Neural Networks (CNN)

A. Autoencoders

An Autoencoder (AE) is a neural network that is trained to regenerate its input vector as its output. It has a hidden layer 'h' that learns the latent representation of input vector (code) in a different feature space with smaller dimensions [4]. Both input and output layers contain N nodes, and hidden layer contain K nodes. If $K < N$ then AE is called under completed. Hidden layer of AE is known by different names in literature including bottleneck, discriminative, coding or abstraction layer. We will stick to the name "bottleneck" and use these names interchangeably if required. Learning task in an under-complete AE forces it to capture most significant features of training data at bottleneck layer so that the input can be regenerated at the output layer. This is achieved by minimizing a loss function $L(x, g(f(x)))$ penalizing dissimilarity of $g(f(x))$ from training data x. In practice, AEs are created from multiple layers where the outputs of preceding layer are connected to inputs of the following layer.

Let W^l and b^l denotes the parameters for each layer where W and b are weights and bias associated with connection between layer l and layer $l+1$, the encoding step for stacked AE is defined by running the encoding step of each layer in forward order $a^l = f(z^l)$ where a denotes activation outputs, $f(.)$ denotes activation function and z denotes total weighted sum of inputs in layer l as shown below:

$$z^{l+1} = W^l.a^l + b^l$$

Decoding step of AE is performed by running the decoding stack in reverse order $a^{n+l} = f(z^l)$ as shown below:

$$z^{n+l+1} = W^{n-l}.a^{n+l} + b^{n-l}$$

where a^n denotes activations of bottleneck layer.
Deep AEs offer many advantages. One major benefit of deep AE stems from universal approximator theorem. [37]. Universal approximator theorem states that a feed-forward neural network with at least one hidden layer can represent an approximation of any function to an arbitrary degree of accuracy which means a deep AE with more than one encoder layer can approximate any mapping from input to bottleneck arbitrarily well. Adding Depth has the effect of exponential reduction in computation cost and amount of training data needed for representing many functions [4]. Different AE structures are described in the literature, but we will discuss the AEs relevant to our study.

1) Sparse Autoencoders: Autoencoder, whose training criterion involves both reconstruction error and sparsity penalty $\Omega(h)$ on the bottleneck layer, is known as Sparse Autoencoder. Sparse AE can be represented as:

$$L(x, g(f(x))) + \Omega(h)$$

Sparsity helps to thwart overfitting of an AE which would otherwise act as an identity function for training data. The penalty term is usually a regularizer added to bottleneck layer.

2) Denoising Autoencoders: Proposed by Vincent et al. [10], denoising autoencoder (DAE) receives a corrupted data instance as input and is trained to predict the original, uncorrupted version of instance as its output. Instead of usual AE loss function, DAE minimizes objective function $L(x, g(f(\bar{x})))$, where $\bar{x}$ represents a corrupted version of x using some noise. DAE must undo this corruption rather than simply replicating their input at output and in doing so it captures only the most significant features of training data. A Corrupting function $C(\bar{x}|x)$, which denotes a conditional distribution over corrupted data instances $\bar{x}$ given original data instances x, is used to generate inputs and DAE learns a reconstruction distribution $P_{re}(x|\bar{x})$ estimated from training data $(x, (\bar{x}))$. DAE can be conceptualized as performing stochastic gradient decent on following expectation:

$$-\mathbb{E}_{x\sim\hat{p}_{data}}\mathbb{E}_{x\sim C(\bar{x}|x)} log p_{decoder}(x|h = f(\bar{x}))$$

Additional benefit of adding noise is reduction in overfitting of models generated by DAE.

3) Contractive Autoencoder: Contractive Autoencoders were proposed by Rifai et al. [12]. Contractive Autoencoders (ContAE) learn representations which are robust towards small changes in training data. This robustness to small changes is achieved by imposing a penalty term based on Frobenius norm of the Jacobian matrix for the encoder activations with respect to the input samples. According to [12], the penalty term compresses the localized space and this contraction in localized feature space yields robust features on the activation layer. The penalty terms also aid in learning representations corresponding to a lower-dimensional non-linear feature space which are more aligned to local directions of variation dictated by data, while invariant to the majority of directions orthogonal to the feature space. Loss function of ContAE is given as follows:

$$\mathbb{T}_{CAE}(\theta) = \Sigma_{x\in D_n}(L(x, g(f(x))) + \lambda||\mathbb{J}_f(x)||_F^2$$

in which $||\mathbb{J}_f(x)||_F^2$ is the Frobenius norm of the jacobian matrix, which is the sum of squares over all elements inside the matrix. Frobenius norm is regarded as the generalization of euclidean norm and represented by following equation.

$$||\mathbb{J}_h(X)||_F^2 = \Sigma_{ij}(\delta h_j(X)/\delta X_i)^2$$

4) Convolutional Autoencoders: Convolutional Autoencoders (ConvAE) were proposed by Masci et al. [11]. ConvAEs learn non-trivial features using simple stochastic gradient descent (SGD), and discover good initializations for Convolutional Neural Networks (CNNs) that avoid the numerous distinct local minima of highly non-convex objective functions arising in different deep learning problems. Fully connected AEs ignore the 2D representation of input which introduces redundancy in parameters, forcing each feature to be global. However, a different approach is to discover localized features that repeat themselves all over the input. ConvAEs differ from other AEs as their weights are shared among all locations of the input preserving spatial locality. The latent representations generated by ConvAEs are more sensitive to transitive relations of features and capture the semantics ignored by other AE structures. For input x shaped as mono channel, the latent representations of kth feature map is given by $h^k = \sigma(x * W^k + b^k)$ where σ is activation function and $*$ denotes 2D convolution whereas x, W and b denotes input, weights and bias respectively. Reconstruction of ConvAE is obtained using following equation:

$$y = \sigma(\Sigma_{k\in H} h^k * \bar{W}^k + c)$$

Where "H" represents the collection of latent feature maps; $\bar{W}$ represents the flip operation over both dimensions of the weights. [11] proposed plain SGD as optimizer and Mean Squared Error as objective function for their ConvAE structure.

B. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [8] are neural network architectures specially crafted to handle high dimensional data with some spatial semantics. Examples of such data include images, video, sound signals in speech, character sequence in the text, or any other multi-dimensional data. In all of the abovementioned cases, using fully connected networks becomes cumbersome due to larger feature space. CNNs are preferred in such cases because of awareness of spatial layout of input, specific local connectivity, and parameter sharing schemes [38].

For CNN, the input x consist of a multi-dimensional array called tensor. The core computational block of CNNs are convolution *Conv* and *pooling* layers. A *Conv* layer takes input as a tensor and convolves it with a set of filters (kernels) to produce output tensor. For a single filter k of dimension d_k, h_k and w_k, convolution is performed by sliding filter k overall spatial positions of the input tensor and calculating dot product between input chunk and filter k to produce a feature map. Since *Conv* layer contains set of filters, it produces a feature map for every filter, and these feature maps are stacked together to produce output tensor. Formally, assuming input shaped as greyscale images, the operation of convolution layer is specified by following steps [38]:

1) Accepts a tensor of size $D_1 * H_1 * W_1$ and requires following four hyper-parameters:
   - Number of filters K
   - Spatial dimensions of filters F
   - Stride with which they are applied S and
   - Size of zero-padding if padding is enabled
2) Conv layer produces an out tensor of size $D_2 * H_2 * W_2$ where $D_2 = K, W_2 = (W_1 - F + 2P)/S + 1$ and $H_2 = (H_1 - F + 2P)/S + 1$
3) The number of parameters in each filter is $F * F * D_1$ for a total of $(F * F * D_1) * K$ weights and K biases

Purpose of *pooling* layer is to control overfitting by decreasing the size of representation with a fixed down-sampling technique (max-pooling or mean-pooling) without any weights. Pooling layers operate on each feature map separately to reduce its size. A typical setting is to use max-pooling with 2x2 filters with a stride of 2 to downsample the representation precisely by half in both height and width.

C. LSTM

Long Short term Memory (LSTM) is a special Recurrent neural network (RNN) architecture. An RNN is a connectivity pattern that perform computations on a sequence of vectors $x_1, \cdots, x_n$ using a recurrence formula of the form $h_t = f_\theta(h_{t-1}, x_t)$, where f, an activation function and θ, a parameter, are used at every timestamp to process sequences with arbitray lengths. The hidden vector h_t is called state of the RNN and it sort of provides a running summary of all vectors x till the time-step and this summary is updated based on current input vector x_n. Vanilla RNNs use linear combinations of x_t, h_{t-1} which are weak form of coupling between inputs and hidden states [39]. Formal form of LSTM is shown below:

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ tanh \end{pmatrix} W \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix}$$

TABLE II: Summary of implemented Autoencoders

Layer Name	Output Shape	No. of Trainable Parameters
Input Layer	(Batch-size, 41)	0
FC1 (H1)	Batch-size, 32	$32 * (41 + 1) = 1344$
FC2 (H2)	Batch-size, 24	$24 * (32 + 1) = 792$
Bottleneck	Batch-size, 16	$16 * (24 + 1) = 400$
FC3 (H3)	Batch-size, 24	$24 * (16 + 1) = 408$
FC4 (H4)	Batch-size, 32	$32 * (24 + 1) = 800$
Output	Batch-size, 41	1353

In equation above, $sigmoid$ and $tanh$ functions are applied element-wise. The tensor W has dimensions $[4H * (D + H)]$. The vectors $i, f, o \in R^H$ intuitively resemble binary gates controlling operations of memory cell. The vector $g \in R^H$ is used to additively modify the memory contents, which allows gradients to be distributed equally during backpropagation.

IV. Methodology

This section describes the architectures of Deep Neural Networks used in this study and methodology of their usage for developing Anomaly detection models.

A. Autoencoders

In our experiments all AEs, except ConvAE, share the same architecture as shown in Fig 1. In figure 1, Model depicted under solid border line shows the AE model. AEs are trained using NSLKDDtrain+ dataset. After training of AE, Bottleneck layer reduces the dimensionality from 41 to 16 dimensional feature space while keeping feature information required for reconstruction. Bottleneck representations of input are fed to an MLP for anomaly detection. The MLP model is shown by dotted border in figure 1 which contains the encoder part of AE after training and a fully connected layer. The output layer contains one sigmoid unit to classify the input. If the output < 0.5 then input instance is classified as normal and vice versa.

1) Vanilla AE is built as described above.
2) For Sparse AE, $L1$ regularization is attached to the input of bottleneck layer to ensure acquisition of unique statistical features from the Dataset. After training, the acquired unique features are fed to MLP for classification as discussed above.
3) For Denoising AE (DAE), input is corrupted using Gaussian noise. The level of input corruption is maintained at 50%. Like earlier AEs, weights of DAE are frozen after training. MLP and output layer is appended to frozen encoder part of DAE to get an MLP classification model.
4) For Contractive AE, only the loss function is changed to contractive loss in accordance with the recommendations of [12] and remaining structure of AE and MLP is same.
5) Output shapes and no. of trainable parameters for trained AEs and trained IDS models are shown in table II and III
6) Architecture of implemented ConvAE is depicted in fig 2 and discussed below.

ConvAE accepts input in form of images. Each NSLKDD training record is shaped as $32x32$ greyscale image. At first,

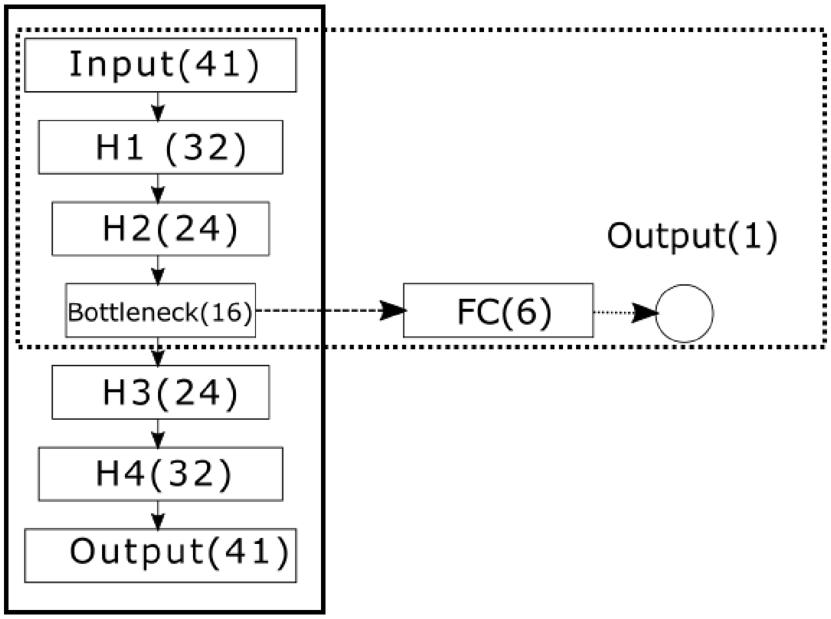

Fig. 1: Architecture of Implemented Autoencoders for Anomaly Detection

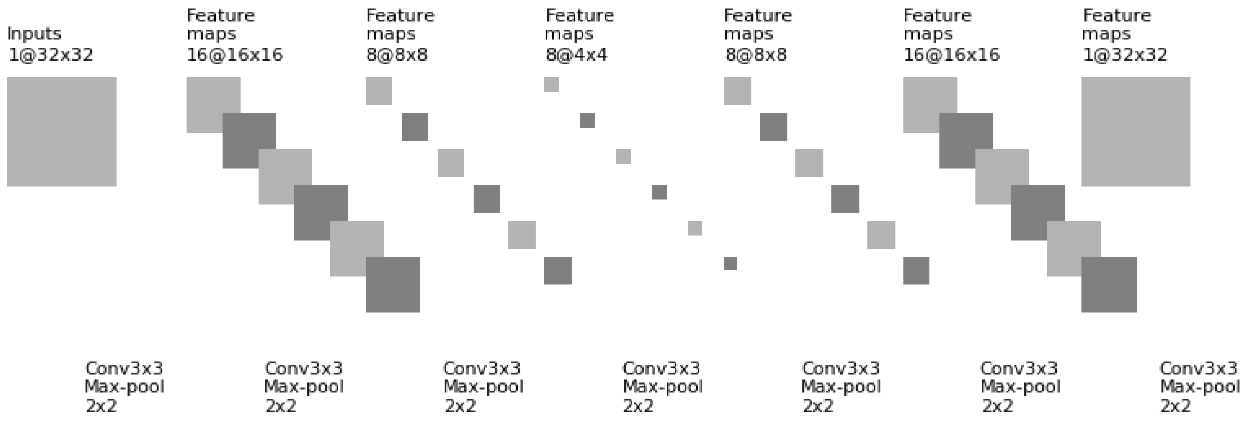

Fig. 2: Architecture of Implemented Convolutional Autoencoder for IDS

the idea of converting a 41 feature input to a 32x32 2D array seems absurd but this approach has its merits. Arranging input features as 2D array helps to discover interesting localized relationships between these features that repeat themselves all over the input. ConvAEs differ from other algorithms as their weights are shared among all locations of the input, preserving spatial locality. The latent representations generated by ConvAE for classification are more sensitive to transitive relationships of features and help to learn high level relationships between global features which would otherwise be ignored by other classifiers. As ConvAEs make use of GPUs for training, the training time of network with 2D input is not much different than a conventional SVM or K-NN classifier. The evidence of abovementioned observation is presented in results section where training and testing times of models are discussed.

TABLE III: Summary of implemented IDS Models based on AEs

Layer Name	Output Shape	No. of Trainable Parameters
Input Layer	(Batch-size, 41)	0
FC1 (H1)	Batch-size, 32	$32 * (41 + 1) = 1344$
FC2 (H2)	Batch-size, 24	$24 * (32 + 1) = 792$
Bottleneck	Batch-size, 16	$16 * (24 + 1) = 400$
FC6 (MLP)	Batch-size, 6	$6 * (16 + 1) = 102$
Output (MLP)	Batch-size, 1	$1 * (6 + 1) = 7$

For converting network flow dataset to corresponding image dataset, we need to create a mapping $F : \theta \rightarrow I$, where I represents image Dataset corresponding to θ and $\theta = (\phi_n)_{n=1}^{N}$is the preprocessed network flow Dataset. To achieve image representation I corresponding to each training instance, vector $v1$ of length 41 is generated from the preprocessed entries of dataset features and replicated 3 times to generate a corresponding vector of 123 features which is converted to a vector $\bar{v}$ of 128 after concatenating first 5 features. For each training/testing instance $\bar{v}$ is replicated to generate corresponding 32x32 greyscale representation. After transforming $\theta \rightarrow I$, the label data is preprocessed as per requirements of two-class structure. The result of label transformation is represented by $y = y_n \in (0,1)^M$ where M denotes the total number of classes. The entry of vector y_n is zero if corresponding image belongs to normal traffic and 1 otherwise. Both Test Datasets NSLKDDTest+ and NSLKDDTest21 are also subjected to same preprocessing.

B. Deep Convolutional Neural Networks

Like ConvAE, Deep Convolutional neural network (DCNN) also requires input in the form of images, hence Datasets were subjected to same preprocessing as that of ConvAE. Architecture of DCNN Model implemented for anomaly detection in this study is shown in Figure 3. The model consist of input layer, three convolution and subsampling pairs, three fully connected layers followed by an output layer consisting of single sigmoid unit. A dropout layer (not shown in figure) is placed between flattened model and first fully connected layer FC1. Dropout Layer, introduced by Srivastav et al. [40], serves as regularization layer. It randomly drops units from the DCNN along with their weights during training time. This has the effect of training an ensemble of neural networks where each member of ensemble is a subset of original neural network. At test time, it is easy to approximate predictions of all ‘thinned’ subsets by simply using an “unthinned” original network with smaller weights. The selected hyper-parameters include softsign activation, He-normal kernel initialization [41], Adadelta optimizer [42] with batch size of 64 instances. Additional hyper-parameters of DCNN included output layer of single sigmoid unit, drop-out rate of 0.5 and zero-padding at each convolution layer input.

C. LSTM

In LSTM IDS Model, each record was processed as single member sequence of 41 dimensional vector to 32 LSTM units. Two Dense layers with ten (10) and one neuron respectively were attached with LSTM outputs to make predictions for two class problem. First Dense layer used RELU activation while classification layer with single unit used sigmoid activation to make predictions. A drop-out layer was introduced between LSTM output and MLP input to thwart over-fitting. LSTM IDS Model was trained on combined Training Dataset for 15 epochs.

V. Implementation

This section describes the experimental setup, preprocessing of datasets and implementation details of deep and conventional models implemented for experiments.

A. Experimental Setup

Hardware setup used for implementing proposed models included:

- CPU : Intel Xeon E-1650 Quad Core
- RAM : 16 GB
- GPU : nVidia GTX 1070 with 1920 CUDA cores and cuda 8.0

B. Preprocessing

A network flow, ϕ, is an ordered set of all packets $\pi_1, \cdots, \pi_n$ where $\pi_i = \{t_i, S_i, Di, s_i, d_i, p_i, f_i\}$ represents a packet such that:

1) $\forall \pi_i, \pi_j \in \phi, p_i = p_j$
2) $\forall \pi_i, \pi_j \in \phi, (S_i = S_j, D_i = D_j, s_i = s_j, d_i = d_j) \wedge (S_i = D_j, S_j = D_i, s_i = d_j, d_i = s_j)$
3) $\forall \pi_{i \neq n} \in \phi (t_i \leq t_{i+1}) and (t_{i+1} - t_i \geq \alpha)$

$where t_i, S_i, Di, s_i, d_i, p_i, f_i$ represents time-stamp, source IP address, destination IP address, source port, destination port and protocol and TCP flags respectively. IDS problem is approached as two-class problem where network flows are either anomalous or normal. Training dataset is prepared by combining $NSLKDDTrain20\%$ and $NSLKDDTrainplus$ which collectively provide 1,51,165 network flow records. NSLKDD has 41 features like its predecessor KDDCUP99 and we have used all 41 features. Out of 41 features, 3 features ‘protocol-type’, ‘service’ and ‘flag’ are symbolic features which require conversion into quantitative form before they can be consumed by DNNs. Different techniques [43],[44], [45] and [46] have been proposed in literature for encoding symbolic features to quantitative features. We studied the impact of different category encoding schemes on classification accuracy of NSLKDD dataset using a conventional classifier. For this purpose we chose Decision-Tree algorithm due to its time efficiency. Impact of different encoding schemes on dimensionality of dataset, training time and accuracy of trained model are shown in Table IV.

In Table IV, Dimensionality shows the number of new features inserted by encoding algorithm in each instance during encoding of three symbolic features. Average scores show the training accuracy of selected Decision-Tree classifier while using a particular encoding scheme. Based on the performance of symbolic feature encoders, we chose LeaveOneOutEncoding proposed by [46].

In general, learning algorithms benefit from standardization of the Dataset. Since different feature vectors of NSLKDD Dataset contained different numerical ranges, we applied scaling to convert raw feature vectors into more standardize representation for DNNs. As Datasets contained both normal and anomalous traffic, to avoid the negative influence of sample mean and variance, we used median and interquartile range (IQR) to scale the data for better results. We removed

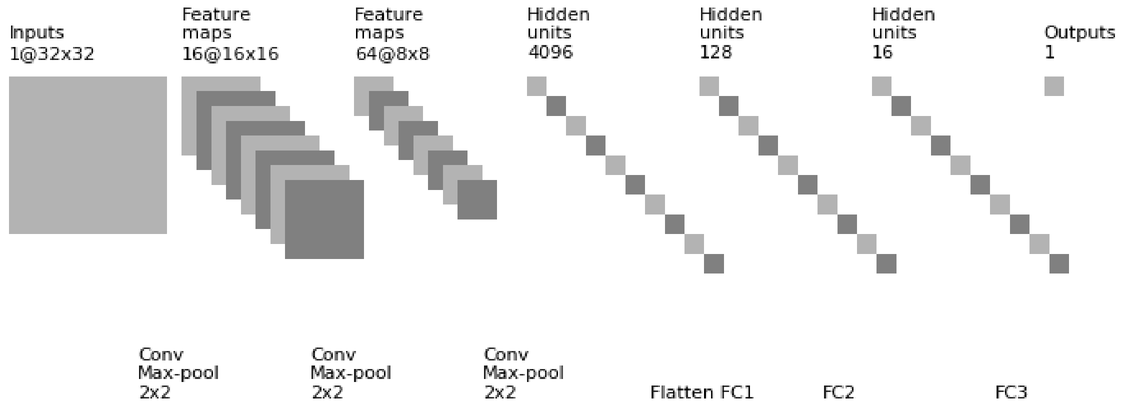

Fig. 3: Architecture of Implemented Deep Convolutional Neural Network model for IDS

the median and scaled the data according to IQR.

C. Implementation of DNN Models

Software toolchain used to implement all DNNs consist of Jupyter development environment using Keras 2.0 [47] on Theano [48] backend and nVidia cuda API 8.0 [49]. Both Training and testing datasets were manipulated in the form of numpy arrays. Python Scikit-learn [50] library was used for various ML related tasks. Figures and graphs were created using python matplotlib and seaborn libraries.

D. Implementation of Conventional Models

For comparisons, we used Scikit-learn [50] implementations of Binary classification algorithms to train conventional models. These models were trained on unraveld version of Training Datasets. Classification algorithms used to train conventional models include Extreme Learning Machine [51] with Generalized hidden layer proposed by [52], RBF SVM, Decision Tree (J48) with 10 node depth, Naive Bayes, Random-Forest with 10 J48 estimators, Quadratic Discriminant Analysis and Multilevel perceptron (MLP).

VI. Results and Evaluations

This section presents results and evaluations of implemented DNN based IDS models. Comparisons of DNN IDS models are provided with each other and with different conventional models. We used prominent metrics to evaluate classification quality of implemented DNNs, which include Receiver Operating Characteristic (ROC), Area under Curve (AuC), Precision-Recall Curve, Accuracy on Test Datasets and mean Average Precision (mAP). These evaluation metrics are computed using confusion matrix which presents four measures as follows:

- True Positive (TP): if an anomaly is classified by model as an anomaly, result is accepted as TP
- False Positive (FP): if a normal instance is classified by model as an anomaly, result is accepted as FP
- True Negative (TN) : if an anomaly is classified by model as normal instance, result is accepted as TN
- False Negative (FN): if a normal instance is classified by model as normal instance, result is accepted as FN

In following subsections, we give a brief introduction of relevant quality metric and present the results for all implemented models. Results are presented in the form of graph plots to allow ease of comparisons.

A. Receiver operating Characteristics (RoC)

RoC is a plot of False positive rate (FPR) against True positive rate (TPR) of binary classifiers. FPR is defined as $FP/(FP+TN)$. It corresponds to the proportion of negative data points mistakenly predicted positive to all negative data points. TPR, also called sensitivity or recall, is defined as $TP/(TP + FN)$. TPR corresponds to the proportion of positive data points that are correctly predicted positive to all positive data points. RoC shows a trade-off between sensitivity and specificity of classifier. The closer the RoC curve is to top-left border, the better the quality of predictions by the prediction model and vice versa. RoC curves of implemented DNN models for both NSLKDDTest+ and NSLKDDTest21 dataset are shown in figure 4 and figure 5 respectively. Comparison of RoC curves for both DNN and conventional ML models is shown in figure 6 and figure 7.

B. Area under ROC Curve

Area under RoC Curve (AuC) is a measure of how well a binary classifier can perform predictions of labels. The AuC of a classifier is equal to the probability that the classifier

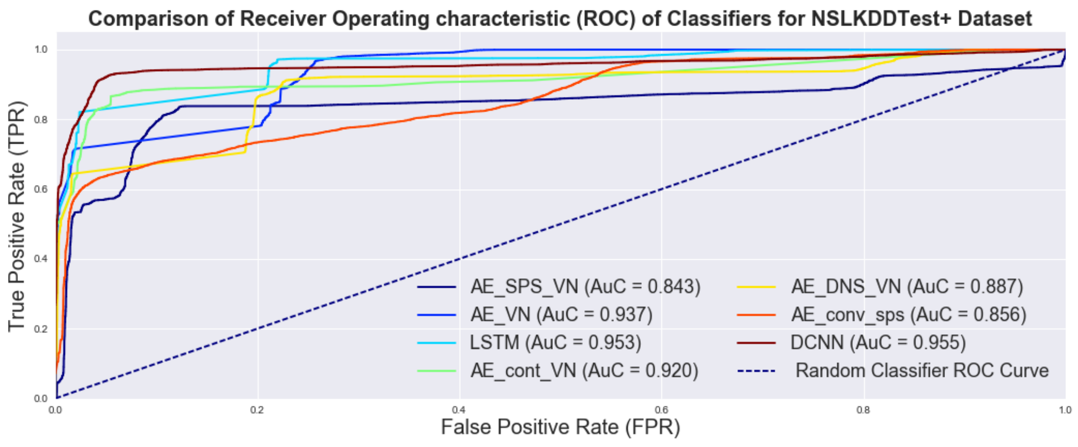

Fig. 4: Comparison of RoC Curves of Deep Neural Network IDS models for NSLKDDTest+ Dataset

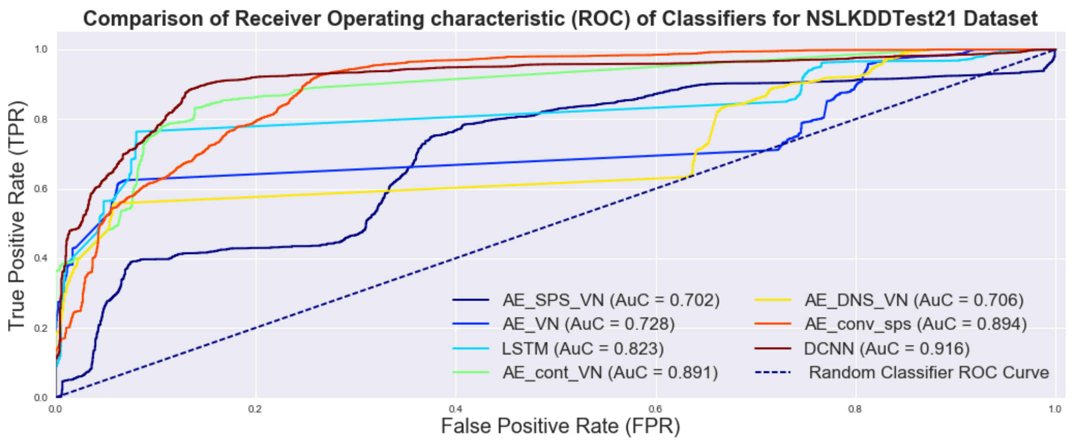

Fig. 5: Comparison of RoC Curves of Deep Neural Network IDS models for NSLKDDTest21 Dataset

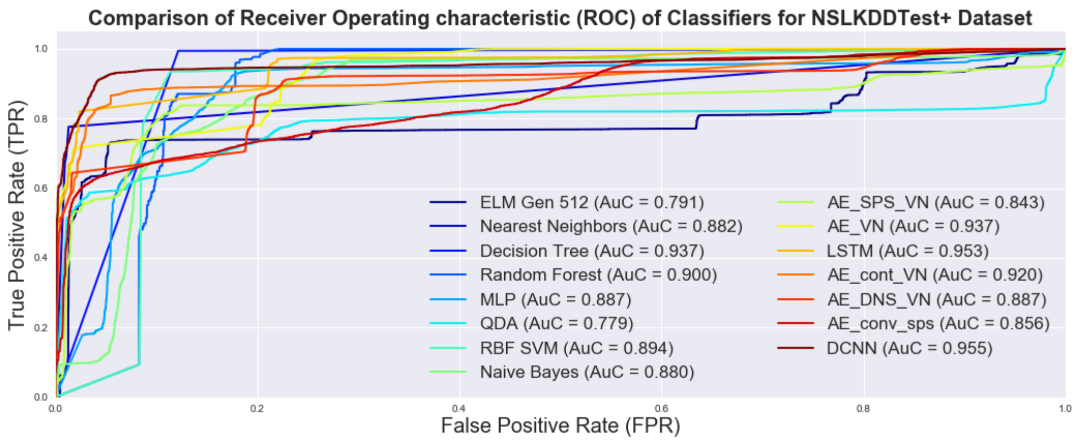

Fig. 6: Comparison of RoC Curves for both Deep and Conventional IDS Models for NSLKDDTest+ Dataset

TABLE IV: Impact of Different Category encoders on Training Results

Encoding Scheme	Dimensionality	Average Score	Score StDev	Training Time (Seconds)
BackwardDifference	81	0.962100	0.002338	9.355746
BinaryEncoder	13	0.962000	0.002265	9.008052
HashingEncoder	8	0.919100	0.002104	20.244040
HelmertEncoder	81	0.962000	0.002336	9.699760
OneHotEncoder	84	0.962075	0.002349	8.856222
OrdinalEncoder	3	0.962125	0.002345	8.188101
SumEncoder	81	0.962050	0.002342	9.365923
PolynomialEncoder	81	0.961975	0.002319	9.985032
BaseNEncoder	13	0.962000	0.002275	10.333861
LeaveOneOutEncoder	3	0.962175	0.002352	8.422500

will rank a randomly chosen positive (Anomalous) record higher than a randomly chosen negative (Normal) one. A perfect binary classifier has $Auc = 1$ and greater value of AuC shows better performance. Any AuC value less than 0.5 indicates poor performance of classifier. The AuC values for both NSLKDDTest+ and NSLKDDTest21 datasets are shown in 6 and figure 7. Top 5 AuC scores for both test datasets are shown in Table V

C. Accuracy

Accuracy results of both Deep and Shallow models are shown in figure 8. In DNN models, LSTM model delivered top accuracy of 89% and 83% for both NSLKDDTest+ and NSLKDDTest21 datasets respectively. DCNN remained runner-up in DNN models with 85% accuracy in NSLKDDTest+ while for NSLKDDTest21 runner-up turned out to be ConvAE. From conventional models, Decision-Tree, SVM and k-NN had a tie at 82% for NSLKDDTest+, while Decision Tree outperformed the two conventional contender models in NSLKDDTest21 dataset with accuracy of 68%. Overall best accuracy was delivered by LSTM with accuracy score of 89% and 83% respectively for NSLKDDTest+ and NSLKDDTest21 datasets. The sharp difference in Accuracies between NSLKDDTest+ and NSLKDDTest21 in all models is due to the fact that NSLKDDTest21 contains records for attack types not available in other NSLKDD train and test Datasets. These attack types include processtable, mscan, snmpguess, snmpgetattack, saint, apache2, httptunnel, back and mailbomb as mentioned earlier. This means that trained models never had the opportunity to see these attacks during training as they were not available in training data.

D. Precision-Recall Curve and mAP

Precision is defined as a measure of relevancy of results, while recall provides us a measure of how many genuinely relevant results are returned. High scores for both show that the model is returning accurate results (high precision), while also returning the majority of positive results (high recall). Each classifier exhibits a trade-off between precision and recall. Due to the fact that individually both Precision and Recall provide only a puzzle piece of classifier performance, they are combined to form Precision-Recall curve which presents the relationship between them in more meaningful manner. The stair-step nature of Precision-Recall curve provides insight into the relationship between precision and recall. A small change in the threshold at the edges of stair-step considerably reduces precision with only a small increase in recall.
Figure 9 and 10 depicts precision-recall curves (PRC) and mean Average Precision (mAP), shown as area under precision-recall curve in legends section, of DNN models for both test Datasets. Mean average precision (mAP) summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with differential increase in recall used as the weight. mAP for all tested models is shown in legends section of Figures 9,10, 11 and 12. Except SparseAE, all DNN models showed very good results. In NSLKDD+, both LSTM and DCNN model share Top position with mAP scores of 97% while DCNN showed marginally improved performance for NSLKDD21 with 98% score. Three models including ContAE, ConvAE and LSTM achieved 97% mAP score for NSLKDD21. Figure 11 and 12 shows PRC and mAP performance of all models. Top six mAP scores are shown in Table VI.

E. Test and Train Timings

In this subsection, we provide the train and test timings of models used in this study. For DNNs, GPU is used as training and testing device while conventional models were trained and tested using CPU. In DNNs, ConvAE proved to be the most expensive algorithm because the training time included both Autoencoder model training and MLP classification model training. Collectively ConvAE IDS model took approximately 367 seconds on GPU. DCNN and LSTM models took 109 and 208 seconds respectively. Smallest training time from DNN models was that of Sparse Autoencoder but it did not show comparable results. SVM with RBF kernel proved to be the most expensive model among conventional IDS models and took approximately 314 seconds. Fastest among conventional category was Random-Forest closely followed by Decision Tree. With smallest training time, Decision-tree model showed remarkable results and performed comparable to other complex models. Remaining conventional models took each under 100 seconds for training. Training times of all models used in this study are shown in 13. Evaluation times of all models used in this study are shown in Figure 14 for both NSLKDDTest+ and NSLKDDTest21 datasets. Overall longest time to evaluate complete dataset was approximately 8 seconds shown by k-NN based IDS model. Both best performing DNN models i.e. DCNN and LSTM took approximately 2 and 4 seconds for NSLKDDTest+ and 1 second for NSLKDDTest21

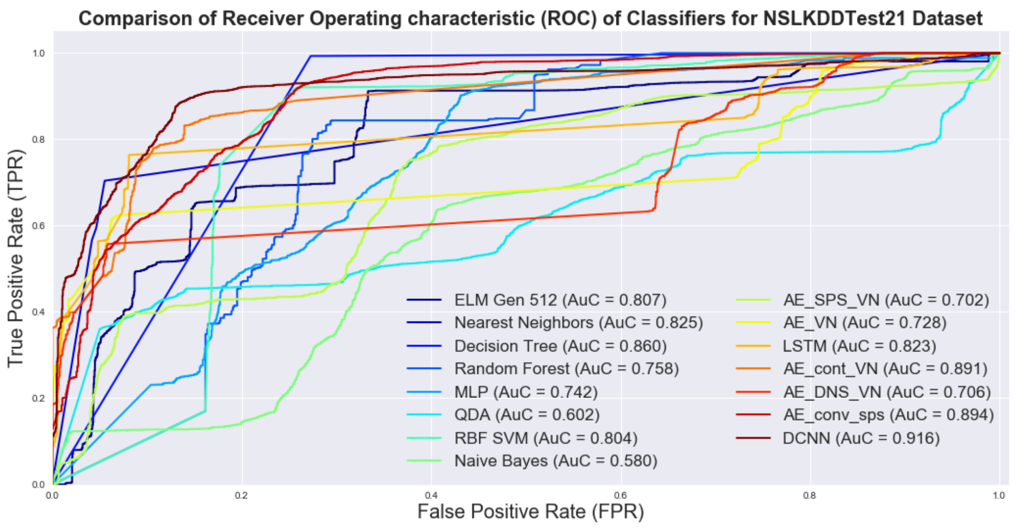

Fig. 7: Comparison of RoC Curves for both Deep and Conventional IDS Models for NSLKDDTest21 Dataset

TABLE V: Top 5 Area under RoC Curve results of Models for $NSLKDDplus$ and $NSLKDD21$ Datasets

Model Name	AuC for NSLKDDPlus	Model Name	AuC for NSLKDD21
DCNN	0.955	DCNN	0.916
LSTM	0.953	Convolutional AE	0.894
AutoEncoder	0.937	Contractive AE	0.891
Decision Tree	0.937	Decision Tree	0.860
Contractive AE	0.920	k-NN	0.825

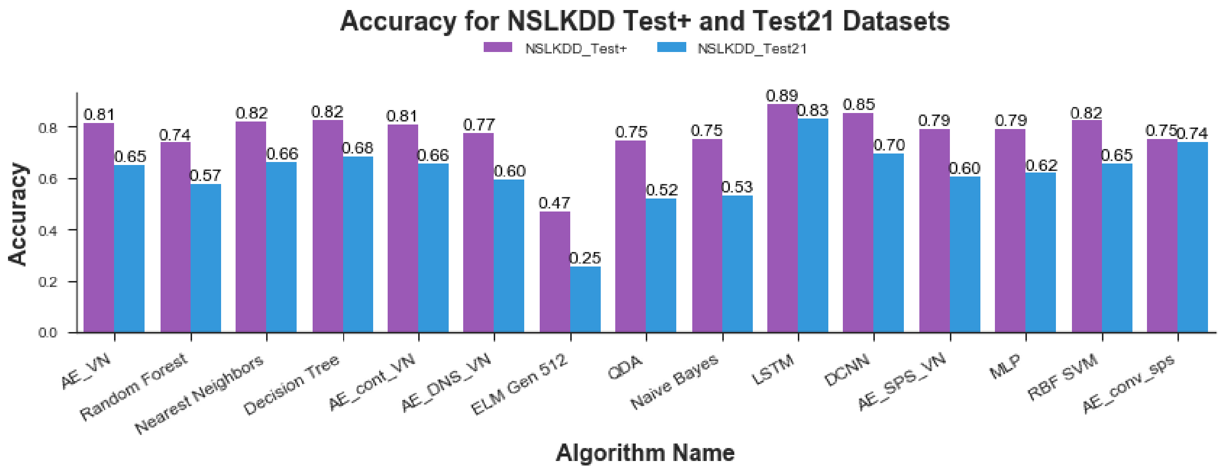

Fig. 8: Comparison of Model accuracies for NSLKDDTest+ and NSLKDDTest21 Datasets

TABLE VI: Top 6 mean Average Precision results from implemented IDS Models

Model	mAP for NSLKDDPlus	Model	mAP for NSLKDD21
DCNN	0.97	DCNN	0.98
LSTM	0.97	LSTM	0.97
Decision Tree	0.96	Convolutional AE	0.97
Contractive AE	0.95	Contractive AE	0.97
k-NN	0.95	k-NN	0.96
AutoEncoder	0.95	Decision Tree	0.95

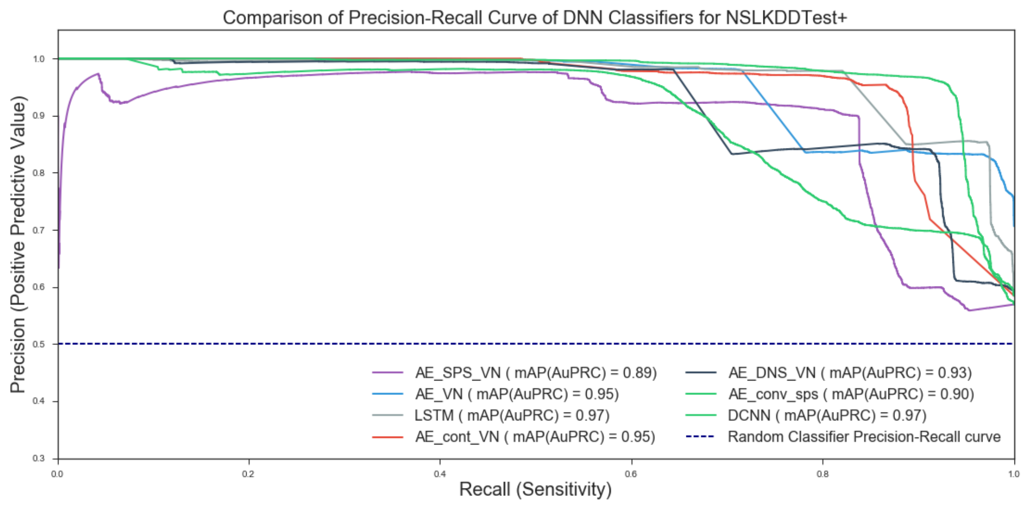

Fig. 9: Precision-Recall Curve and mAP scores of DNN models for NSLKDDTest+ Dataset

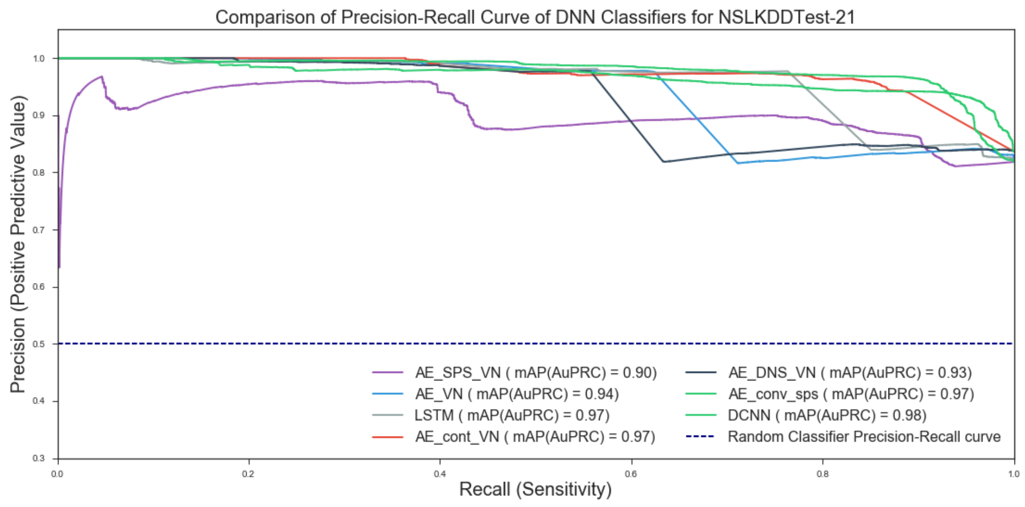

Fig. 10: Precision-Recall Curve and mAP scores of DNN models for NSLKDDTest21 Dataset

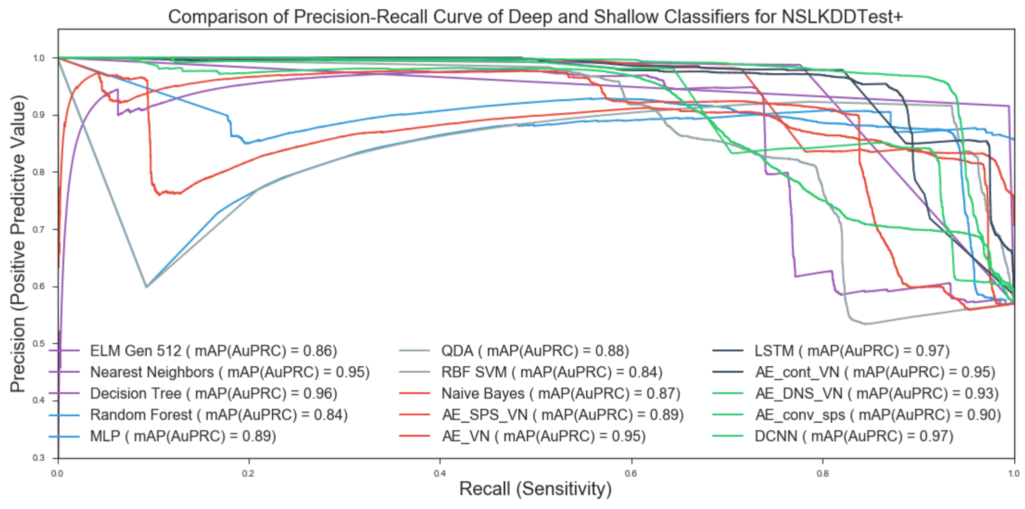

Fig. 11: Precision-Recall Curve and mAP scores of DNN and conventional models for NSLKDDTest+ Dataset

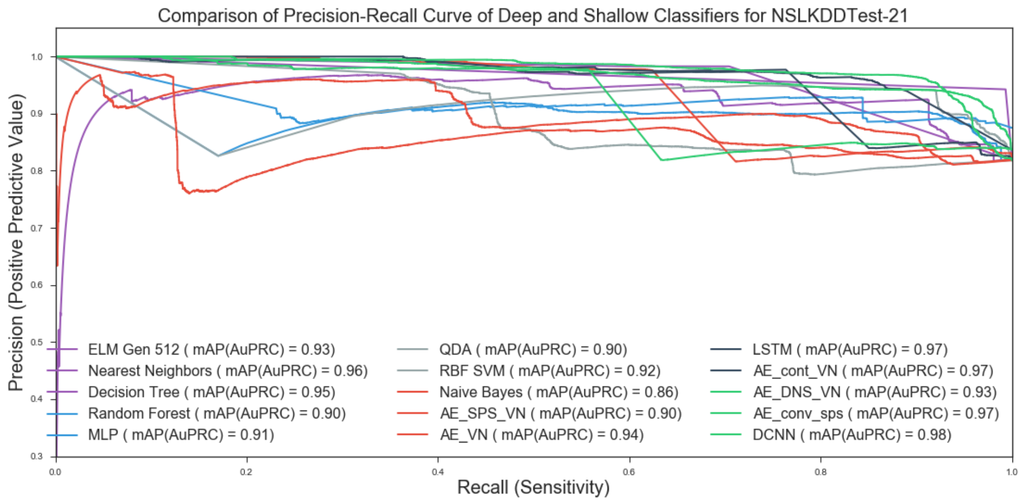

Fig. 12: Precision-Recall Curve and mAP scores of DNN and conventional models for NSLKDDTest21 Dataset

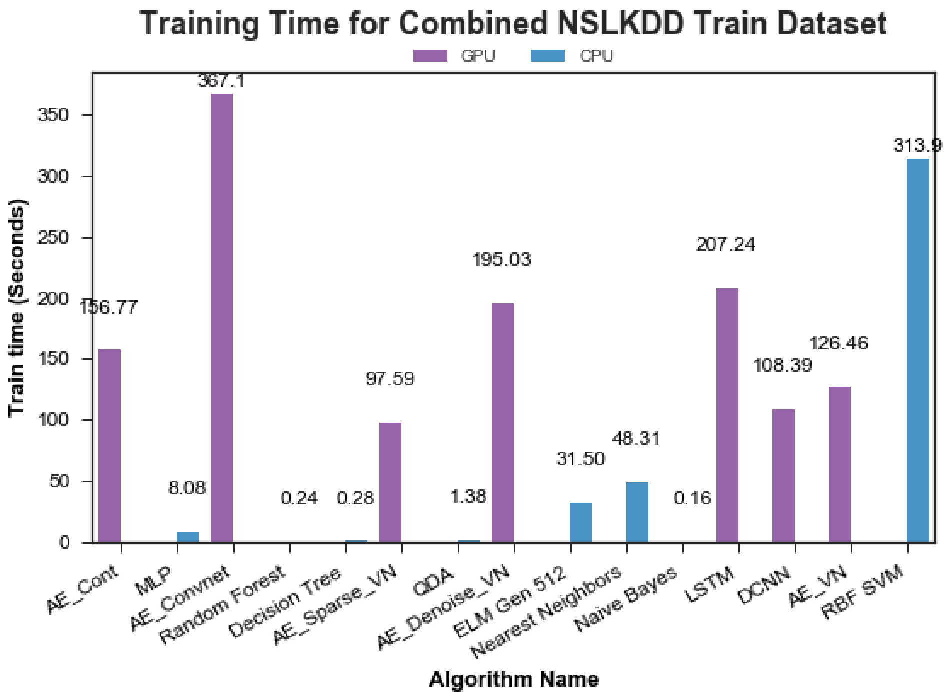

Fig. 13: Training Time in seconds for Different algorithms used in experiments

dataset. Decision Tree based IDS model performed imperceptibly fast during evaluation of both test Datasets.

VII. Conclusion

In this paper, Intrusion Detection models were proposed, implemented and trained using different deep neural network architectures including Convolutional Neural Networks, Autoencoders, and Recurrent Neural Networks. These deep models were trained on NSLKDD training dataset and evaluated on both test datasets provided by NSLKDD namely NSLKDDTest+ and NSLKDDTest21. For training and evaluation of deep models, a GPU powered test-bed using keras with theano backend was employed. To make model comparisons more credible, we implemented conventional ML IDS models with different well-known classification techniques including Extreme Learning Machine, k-NN, Decision-Tree, Random-Forest, Support Vector Machine, Naive-Bays, and QDA. Both DNN and conventional ML models were evaluated using well-known classification metrics including RoC Curve, Area under RoC, Precision-Recall Curve, mean average precision and accuracy of classification. Both DCNN and LSTM models showed exceptional performance with 85% and 89% Accuracy on test dataset which demonstrates the fact that Deep learning

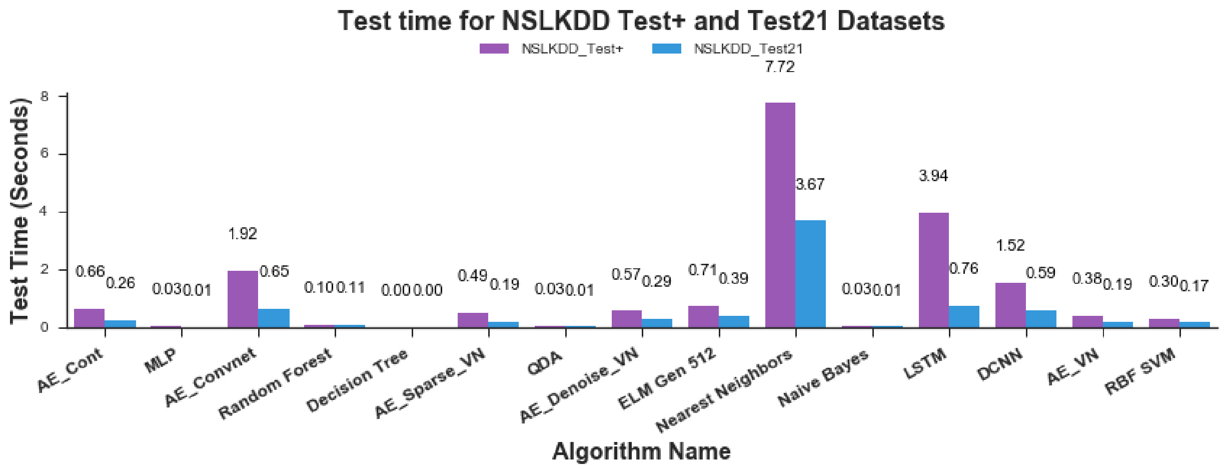

Fig. 14: Test Time for Different algorithms used in experiments for both NSLKDDTest+ and NSLKDDTest21 Datasets

is not only viable but rather promising technology for information security applications like other application domains. Our future research will be directed towards investigating deep learning as feature extraction tool to learn efficient data representations for anomaly detection problem.

References

[1] D. E. Denning, “An intrusion-detection model,” *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/1702202/

[2] M. Luo, L. Wang, H. Zhang, and J. Chen, “A Research on Intrusion Detection Based on Unsupervised Clustering and Support Vector Machine,” in *Information and Communications Security: 5th International Conference, ICICS 2003, Huhehaote, China, October 10-13, 2003. Proceedings*, S. Qing, D. Gollmann, and J. Zhou, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 325–336, dOI: 10.1007/978-3-540-39927-8_30. [Online]. Available: https://doi.org/10.1007/978-3-540-39927-8_30

[3] X. Zhu and A. B. Goldberg, “Introduction to Semi-Supervised Learning,” *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 3, no. 1, pp. 1–130, Jan. 2009. [Online]. Available: http://www.morganclaypool.com/doi/abs/10.2200/S00196ED1V01Y200906AIM006

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[5] M. Minsky and S. Papert, “Perceptrons.” 1969.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[8] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[9] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735

[10] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion,” *Journal of Machine Learning Research*, vol. 11, no. Dec, pp. 3371–3408, 2010. [Online]. Available: http://www.jmlr.org/papers/v11/vincent10a.html

[11] J. Masci, U. Meier, D. Cirean, and J. Schmidhuber, “Stacked convolutional auto-encoders for hierarchical feature extraction,” *Artificial Neural Networks and Machine LearningICANN 2011*, pp. 52–59, 2011. [Online]. Available: http://www.springerlink.com/index/U47243T6702223K4.pdf

[12] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, “Contractive auto-encoders: Explicit invariance during feature extraction,” in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 833–840. [Online]. Available: http://machinelearning.wustl.edu/mlpapers/paper_files/ICML2011Rifai_455.pdf

[13] I. Ghafir, V. Prenosil, and M. Hammoudeh, “Botnet command and control traffic detection challenges: A correlation-based solution,” 2016.

[14] A. Carlin, M. Hammoudeh, and O. Aldabbas, “Intrusion detection and countermeasure of virtual cloud systems-state of the art and current challenges,” *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 6, 2015.

[15] J. Kim, N. Shin, S. Y. Jo, and S. H. Kim, “Method of intrusion detection using deep neural network,” in *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*. IEEE, 2017, pp. 313–316. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7881684/

[16] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A Detailed Analysis of the KDD CUP 99 Data Set,” in *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, ser. CISDA’09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 53–58. [Online]. Available: http://dl.acm.org/citation.cfm?id=1736481.1736489

[17] S. D. Bay, D. F. Kibler, M. J. Pazzani, and P. Smyth, “The UCI KDD Archive of Large Data Sets for Data Mining Research and Experimentation,” *SIGKDD Explorations*, vol. 2, p. 81, 2000.

[18] M. Tavallaee, “An adaptive hybrid intrusion detection system,” Ph.D. dissertation, University of New Brunswick, 2011.

[19] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He, “Fuzziness based semi-supervised learning approach for intrusion detection system,” *Information Sciences*, vol. 378, pp. 484–497, Feb. 2017. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0020025516302547

[20] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, “Autoencoder-based feature learning for cyber security applications,” in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 3854–3861. [Online]. Available: http://ieeexplore.ieee.org/abstract/document/7966342/

[21] Y. Liao and V. Vemuri, “Use of K-Nearest Neighbor classifier for intrusion detection,” *Computers & Security*, vol. 21, no. 5, pp. 439–448, Oct. 2002. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S016740480200514X

[22] M. Khan, A. Ahmad, S. Khalid, S. H. Ahmed, S. Jabbar, and J. Ahmad, “Fuzzy based multi-criteria vertical handover decision modeling in heterogeneous wireless networks,” *Multimedia Tools and Applications*, vol. 76, no. 23, pp. 24 649–24 674, 2017.

[23] S. Mukkamala, G. Janoski, and A. Sung, “Intrusion detection using

neural networks and support vector machines." IEEE, 2002, pp. 1702–1707. [Online]. Available: http://ieeexplore.ieee.org/document/1007774/

[24] P. Laskov, P. Dssel, C. Schfer, and K. Rieck, "Learning Intrusion Detection: Supervised or Unsupervised?" in *Image Analysis and Processing ICIAP 2005: 13th International Conference, Cagliari, Italy, September 6-8, 2005. Proceedings*, F. Roli and S. Vitulano, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 50–57, dOI: 10.1007/11553595_6. [Online]. Available: https://doi.org/10.1007/11553595_6

[25] A. A. Ghorbani, W. Lu, and M. Tavallaee, *Network Intrusion Detection and Prevention*, ser. Advances in Information Security. Boston, MA: Springer US, 2010, vol. 47. [Online]. Available: http://link.springer.com/10.1007/978-0-387-88771-5

[26] A. Solanas and A. Martinez-Balleste, *Advances in artificial intelligence for privacy protection and security*, ser. Intelligent Information Systems. Hackensack, N.J.: World Scientific, 2010. [Online]. Available: http://site.ebrary.com/id/10421991

[27] D. K. Bhattacharyya and J. K. Kalita, *Network anomaly detection: A machine learning perspective*. CRC Press, 2013.

[28] A. Abuarqoub, M. Hammoudeh, B. Adebisi, S. Jabbar, A. Bounceur, and H. Al-Bashar, "Dynamic clustering and management of mobile wireless sensor networks," *Computer Networks*, vol. 117, pp. 62–75, 2017.

[29] M. Tausif, J. Ferzund, S. Jabbar, and R. Shahzadi, "Towards designing efficient lightweight ciphers for internet of things." *KSII Transactions on Internet & Information Systems*, vol. 11, no. 8, 2017.

[30] N. Gao, L. Gao, Q. Gao, and H. Wang, "An Intrusion Detection Model Based on Deep Belief Networks." IEEE, Nov. 2014, pp. 247–252. [Online]. Available: http://ieeexplore.ieee.org/document/7176101/

[31] Z. Wang, *The Applications of Deep Learning on Traffic Identification*. blackhat 2015, 2015. [Online]. Available: https://www.blackhat.com/docs/us-15/materials/us-15-Wang-The-Applications-Of-Deep-Learning-On-Traffic-Identification.pdf

[32] R. C. Aygun and A. G. Yavuz, "Network Anomaly Detection with Stochastically Improved Autoencoder Based Models." IEEE, Jun. 2017, pp. 193–198. [Online]. Available: http://ieeexplore.ieee.org/document/7987197/

[33] M. Z. Alom, V. Bontupalli, and T. M. Taha, "Intrusion detection using deep belief networks." IEEE, Jun. 2015, pp. 339–344. [Online]. Available: http://ieeexplore.ieee.org/document/7443094/

[34] E. Hodo, X. Bellekens, A. Hamilton, C. Tachtatzis, and R. Atkinson, "Shallow and Deep Networks Intrusion Detection System: A Taxonomy and Survey," *arXiv preprint arXiv:1701.02145*, 2017. [Online]. Available: https://arxiv.org/abs/1701.02145

[35] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A Deep Learning Approach for Network Intrusion Detection System," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (Formerly BIONETICS)*, ser. BICT'15. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, pp. 21–26. [Online]. Available: http://dx.doi.org/10.4108/eai.3-12-2015.2262516

[36] L. Bontemps, V. L. Cao, J. McDermott, and N.-A. Le-Khac, "Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network," *CoRR*, vol. abs/1703.09752, 2017. [Online]. Available: http://arxiv.org/abs/1703.09752

[37] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, Jan. 1991. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/089360809190009T

[38] A. Karpathy, "Connecting Images and Natural Language," Ph.D. dissertation, Stanford University, 2016. [Online]. Available: https://pdfs.semanticscholar.org/6271/07c02c2df1366965f11678dd3c4fb14ac9b3.pdf

[39] Y. Wu, S. Zhang, Y. Zhang, Y. Bengio, and R. Salakhutdinov, "On Multiplicative Integration with Recurrent Neural Networks," *CoRR*, vol. abs/1606.06630, 2016. [Online]. Available: http://arxiv.org/abs/1606.06630

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1026–1034. [Online]. Available: http://www.cv-foundation.org/openaccess/content_iccv_2015/html/He_Delving_Deep_into_ICCV_2015_paper.html

[42] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[43] W. Mcginnis, "BaseN Encoding and Grid Search in categorical variables," Jul. 2017. [Online]. Available: http://www.willmcginnis.com/2016/12/18/basen-encoding-grid-search-category_encoders/

[44] ——, "Beyond One-Hot: an exploration of categorical variables," Jul. 2017. [Online]. Available: http://www.willmcginnis.com/2015/11/29/beyond-one-hot-an-exploration-of-categorical-variables/

[45] S. C. Group, "Contrast Coding Systems for categorical variables," Feb. 2011. [Online]. Available: https://stats.idre.ucla.edu/r/library/r-library-contrast-coding-systems-for-categorical-variables/

[46] O. Zhang, "Strategies to encode categorical variables with many categories," Feb. 2017. [Online]. Available: https://www.kaggle.com/c/caterpillar-tube-pricing/discussion/15748#143154

[47] F. Chollet and others, *Keras*. GitHub, 2015. [Online]. Available: https://github.com/fchollet/keras

[48] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[49] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable Parallel Programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1365490.1365500

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[51] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1-3, pp. 489–501, Dec. 2006. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0925231206000385

[52] F. Fernandez-Navarro, C. Hervas-Martinez, J. Sanchez-Monedero, and P. A. Gutierrez, "MELM-GRBF: A modified version of the extreme learning machine for generalized radial basis function neural networks," *Neurocomputing*, vol. 74, pp. 2502–2510, 2011.