Enhanced Network Intrusion Detection using Deep Convolutional Neural Networks

Sheraz Naseer[1,2*] and Yasir Saleem[1]
[1] Department of Computer Science & Engineering, University of Engineering and Technology Lahore, Pakistan
[2] Department of Computer Science, School of Systems and Technology, University of Management and Technology, Lahore, Pakistan
*Corresponding Author: Sheraz Naseer
[e-mail: sheraz.naseer@gmail.com]

Received September 21, 2017; revised December 18, 2017; accepted May 13, 2018; published October 31, 2018

Abstract

Network Intrusion detection is a rapidly growing field of information security due to its importance for modern IT infrastructure. Many supervised and unsupervised learning techniques have been devised by researchers from discipline of machine learning and data mining to achieve reliable detection of anomalies. In this paper, a deep convolutional neural network (DCNN) based intrusion detection system (IDS) is proposed, implemented and analyzed. Deep CNN core of proposed IDS is fine-tuned using Randomized search over configuration space. Proposed system is trained and tested on NSLKDD training and testing datasets using GPU. Performance comparisons of proposed DCNN model are provided with other classifiers using well-known metrics including Receiver operating characteristics (RoC) curve, Area under RoC curve (AuC), accuracy, precision-recall curve and mean average precision (mAP). The experimental results of proposed DCNN based IDS shows promising results for real world application in anomaly detection systems.

Keywords: Network Intrusion Detection, Deep Convolutional Neural Networks, Deep learning, CNN, IDS, Information Security

This research was Not supported by any research grant from the IT R&D program of MKE/IITA, the Korean government [2005-Y-001-04, Development of Next Generation Security Technology].

http://doi.org/10.3837/tiis.2018.10.028 **ISSN : 1976-7277**

1. Introduction

Intrusion detection is the process of monitoring, identifying, analyzing and managing IT infrastructure events which can adversely impact the security of information systems. Efficient and effective Intrusion Detection systems are required by every organization because they assure the reliability of IT infrastructure of the organization and hence the undisrupted business operations. The mainstream of intrusion detection systems (IDS) rely on signature based approach but this method is unable to defend against zero-day attacks due to its reliance on presence of attack signatures for successful detection of attacks.

Denning [1] introduced the idea of developing intrusion detection system by using Artificial Intelligence techniques on security events to identify abnormal usage patterns and intrusions. This idea introduced a new breed of intrusion detection systems (IDS) commonly known as anomaly detection systems which employ machine learning and Data mining approaches. These approaches primarily consist of *supervised, unsupervised* and *semi-supervised* learning to propose solutions for anomaly detection problem

In supervised learning, intrusion detection is posed as a classification problem where detection model is trained using both normal and anomalous data. Unsupervised learning uses unlabeled or untagged data for learning tasks. Most popular unsupervised learning technique is clustering [2] in which the learning algorithm searches for similarities among instances of dataset to build group of instances called clusters. Semi-Supervised Learning (SSL) is a combination of supervised and unsupervised learning approaches. The SSL approach utilizes both labeled and unlabeled data [3] for learning.

Deep Learning is an area of Machine Learning which has the objective of moving Machine Learning closer towards Artificial Intelligence [4] by using neuron like mathematical structures for learning tasks. Neural Networks have been around for five decades [5] and have been gaining and losing the favor of research community. The year 2012 marked the rise of Deep neural networks (DNNs) in computer vision when Alexnet [6] 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. Since then DNNs have been successfully used in multiple disciplines which includes information security.

Application of Deep Neural Networks for solution of Information security problems is relatively new area of research. The main contribution of this study is investigation of Deep Convolutional Neural Networks (DCNNs) for network anomaly detection problem. To the best of our knowledge, DCNNs have never been investigated for the aforementioned application. All experiments in this study are performed on NSLKDD dataset provided by [8]. NSLKDD was derived from KDDCUP99 [9] which was generated in 1999 from the DARPA98 network traffic. Tavallaee et al. [8] discovered some inherent shortcomings in the original KDDCUP99 dataset that would have adversely affected the performance of IDS trained and evaluated on the Dataset. A statistically enhanced version of dataset called NSLKDD was proposed by [8] to counter discovered statistical problems. Some advantages of NSLKDD over KDDCUP99 dataset, as described by [8], include removal of redundant records from training dataset for reducing complexity and bias towards frequent records and introduction of non-duplicate records in testing datasets for unbiased evaluation.

The NSLKDD dataset incorporates 41 input features as in original KDDCUP99 dataset and a class label. Features 1 to 9 represent the basic features which are extracted from TCP/IP

connection without inspecting the payload. Features 10 to 22 consist of contents features generated from payload of TCP segments of packets. Features 23 to 31 are extracted from time based traffic properties while features 32 to 41 contains host based traffic features that are designed to assess attack within interval longer than 2 seconds. A class label is provided with each record, which specifies the status of an instance either as normal or an attack. Original KDDCUP99 dataset listed different types of attacks shown in **Table 1**.

Table 1. 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	

NSLKDD Dataset is available in four partitions. Two partitions namely NSLKDDTrain20p and NSLKDDtrain+ each containing 25,192 and 125,973 training records respectively are provide for training of models and remaining two partitions called NSLKDDTest+ and NSLKDDTest21 are used to evaluate the performance of trained models. Both NSLKDDTest+ and NSLKDDTest21 are comprised of 22,543 and 11,850 instances respectively. NSLKDDTest21 also 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.

In this study, a DCNN is implemented for task of two-class supervised anomaly detection. Fine tuning and hyper-parameter optimization of proposed DCNN was performed using RandomizedGridSearch over configuration space. Classification metrics including accuracy, Area under ROC curve, mean average precision, and precision-recall scores were calculated for comparison of DCNN with well-known classification schemes including SVM, K-nearest neighbor, Decision-Tree, RandomForest, Quadratic Discriminant Analysis (QDA) and Extreme Learning machine (ELM) [10].

Rest of the article is divided into V sections. Section II highlights prominent works related to IDS problem. Section III provides the methodology and architectural design of the system. Section IV sheds light on implementation details including hardware setup and software tool-chain. In Section V, we present results of proposed deep CNN based IDS along with comparisons of results and timing information. This section is followed by section V and VI which describes the conclusion and references of research respectively.

2. Related Work

Many machine learning techniques including supervised, unsupervised and semi-supervised, have been proposed to enhance performance of anomaly detection. Supervised approaches such as *k*-nearest neighbor (KNN) [11], neural network (NN)[12], and support vector machine (SVM) have been extensively used to detect the intrusions [13]. A comprehensive repertoire of anomaly based intrusion detection systems is presented by bhattacharraya et al. in [14]. Ghorbani et al. [15] provided an inclusive review of supervised and unsupervised learning approaches for anomaly detection. Tavallee [16] compared the performance of the NSLKDD

dataset on different classifiers including naive bayes, SVM, and random forests etc. Solanas et al. [17] presented clustering algorithms for anomaly detection. Laskov et al [18] provided comparative analysis of supervised and unsupervised learning techniques with respect to their detection accuracy and ability to detect unknown attacks.
Application of Deep Neural Networks for solution of Information security problems is relatively new area of research. Although DCNNs have not been used for IDS, other DNN structures like Autoencoders (AE), Deep belief Networks (DBNs) and LSTM have been. David et al [19] employed a DBN to classify Malware samples. Gao et al. [20] proposed an IDS architecture based on DBNs using energy based reduced Boltzmann machines (RBMs) on KDDCup99 Dataset. Wang [21] proposed a deep network of stacked auto encoders (SAE) for network traffic identification. A semi-supervised learning based approach with Random weights based NN (NNRw) is used by Ashfaq et al. [22] to implement an IDS architecture using NSLKDD.

It is relevant to mention that we encountered two approaches for evaluation of models in literature. In first approach, authors used training datasets for both training and testing of models using cross-validation mechanisms. The studies using this approach reported very high detection rates, e.g. Kim et al. [23] used a four layer DNN with 100 units for intrusion detection on KDD99 dataset and reported 99% accuracy. Similarly Alwardesh et al. [24] used a DNN model and reported 97.9% accuracy. We believe that this approach is flawed, as given sufficient training, classifiers can be over-fitted to achieve such high rates. The second approach is to train the model on training Dataset without ever exposing the test dataset to model during training and then test the model on testing dataset. We believe that 2nd approach is more useful and all models in this 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 [16], [22] and [25].

3. Methodology

This section describes the preprocessing of dataset, architecture of model and hyper-parameter selection method.

3.1 Preprocessing

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

(1) $\forall \pi_i, \pi_j \in \phi, p_i = p_j$

(2) $\forall \pi_i, \pi_j \in \phi, \left(S_i = S_j, D_i = D_j, s_i = s_j, d_i = d_j\right) and \left(S_i = D_j, S_j = D_i, s_i = d_j, d_i = s_j\right)$

(3) $\forall \pi_{i \neq n} \in \phi (t_i \leq t_{i+1}) and\ (t_{i+1} - t_i \geq \alpha)$

where $t_i, S_i, D_i, s_i, d_i, p_i, f_i$ represents time-stamp, source IP address, destination IP address, source port, destination port, protocol and TCP flags respectively. IDS problem in this study is treated as two-class problem where flows are either anomalous or normal. The training dataset is prepared by combining NSLKDDTrain20p and NSLKDDTrain+ which contains 151,165 training instances. 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 needs to be converted to quantitative data before they can be used by

DCNN. Different techniques [26] [27] [28] [29] [30] 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 Random-Forest algorithm due to its time efficiency. Impact of different encoding schemes on dimensionality of dataset and training time and accuracy of classifier are shown in **Table 2**.

Table 2. Impact of Different Category encoders on Accuracy of NSLKDD training Dataset

Encoding Scheme	Dimensionality	Training Time (Seconds)	Average Training Score	Score StDev
BackwardDifference	81	9.445193	0.961925	0.002291
BinaryEncoder	13	9.234833	0.962050	0.002472
HashingEncoder	8	20.524086	0.918650	0.002197
HelmertEncoder	81	9.418384	0.962100	0.002359
OnehotEncoder	84	8.884236	0.961950	0.002361
OrdinalEncoder	3	8.443738	0.961950	0.002513
SumEncoder	81	9.405340	0.961975	0.002560
PolynomialEncoder	81	9.642599	0.962000	0.002327
BaseNEncoder	13	10.734352	0.961925	0.002342
LeaveOneOutEncoder	3	8.746265	0.962150	0.002444

In **Table 2**, dimensionality shows the number of new features inserted by encoding algorithm in each instance during encoding of three symbolic features. Average Training scores show the training accuracy of selected Random-Forest classifier while using a particular encoding scheme. Based on the performance of encoders, we chose LeaveOneOutEncoding proposed by [26].

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 DCNN. 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 the median and scaled the data according to IQR.

DCNNs accept input in form of images. Each NSLKDD training record from Training Dataset is shaped as 32x32 greyscale image. At first, 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 localized features which repeat themselves all over the input. DCNNs differ from other classifiers as their weights are shared among all locations of the input preserving spatial locality. The latent representations generated by DCNN for classification are more sensitive to transitive relationships of features and help DCNNs to learn high level relationships between global features which would otherwise be ignored by other classifiers. As DCNNs can use GPUs for training, the training time of network with 2D input is not that different from a classical SVM or K-NN classifier. The evidence of abovementioned fact is presented in results section where training and testing times of classifiers are discussed.

For converting network Dataset to corresponding image dataset, our first goal is to create a mapping $F: \Phi \rightarrow I$, where I represents image Dataset corresponding to Φ and $\Phi = \{\phi_n\}_{n=1}^{N}$ 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 $\Phi \rightarrow I$, the label data was preprocessed according to 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 were also subjected to same preprocessing.

3.2 Model Architecture

Proposed IDS approach uses a DCNN with an input layer, 3 pairs of conv-subsample layers, 3 fully connected layers and an output layer with one sigmoid unit. The input plane receives preprocessed NSLKDD training dataset records in the form of 32x32 greyscale images as described earlier.

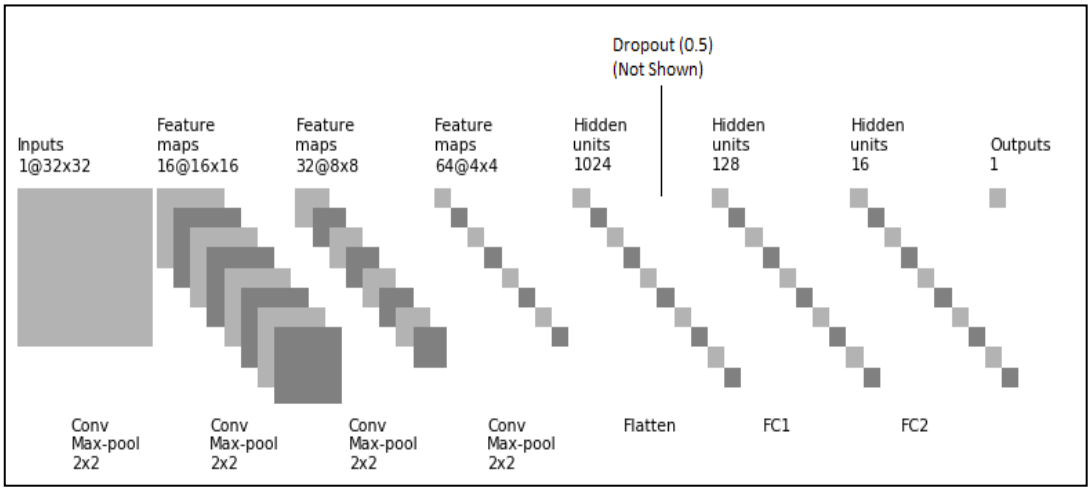

Fig. 1. Architecture of Proposed Deep Convolutional Neural Network (DCNN) for Intrusion Detection

With local receptive fields, earlier layer neurons can extract elementary features which are combined by subsequent CNN layers to form higher-order features. DCNN used in our study is inspired by LeNet-5 [31] but contains heavy modifications in form of hyper-parameter selection and regularization. Modifications include different input shape, different convolution kernels at each convolution layer, different activation and objective functions and a dropout layer to minimize overfitting. Each layer consists of trainable parameters and nodes as described in **Table 3**. We use LeNet-5 nomenclature to name layers of DCNN for description purpose where convolution layers are labeled as Cx, subsampling layers as Sx, dropout layers as Dx and Fully connected layers FCx.

Table 3. Parameters for Individual layers of CNN used for Intrusion Detection

Layer Name : Function	Output Shape	Trainable Parameters (Weights)
C1: Convolution layer with 3x3 kernels and 16 feature maps	16, (32,32)	((2*2)+ 1)*16 = 80
S2: Subsampling layer with 2x2 non-overlapping kernel	16, (16,16)	0 (Sub-sampling)
C3: Convolution layer with 3x3 kernels and 64 feature maps	32, (16,16)	(((2x2)*16)+1)*32 =2080
S4: Subsampling layer with 2x2 non-overlapping kernel	32, (8,8)	0 (Sub-sampling)

C5: Convolution layer with 3x3 kernels and 64 feature maps	64, (8,8)	(((2x2)*32)+1)*64 =8256
S6: Subsampling layer with 2x2 non-overlapping kernel	64, (4,4)	0 (Sub-sampling)
Model Flattening	1024,1	Not Applicable
D1: Dropout layer with 0.5 drop probability		Dropout Layer
FC7: Fully connected layer	128 ,1	(1024 +1)*128=131200
FC8: Fully connected layer	16 ,1	(128 +1)*16=2064
Output: Fully connected layer	1	(16 +1) *1=17

Input is shaped as 32x32 greyscale image. Layer C1 contains 16 feature maps where each feature map is connected to a 2x2 neighborhood region of input image. All convolution layers of DCNN use zero-padding for alignment of input-output shapes. S2 is a subsampling layer with 16 feature maps of size 32x32 and each feature map is connected to a 2x2 non-overlapping region in corresponding feature map is C1 which results in 16 feature maps of 32x32. Layer C3 is convolutional layer containing 32 feature maps and each feature map of C3 corresponds to outputs of a different kernel function operating at several 2x2x16 regions of S2 layer. Layer S4 is a sub-sampling layer of 32 3x3 feature maps and each feature map is connected to a 2x2 non-overlapping region in C3. Layer C5 performs convolution over the output of S4 to generate 64 feature maps. The model is flattened to generate first fully connected layer of 1024 units as shown in **Fig. 1**. A dropout layer D1 is placed between flattened model and first fully connected layer FC7. Layer D1 is a regularization layer using drop out regularization introduced by Srivastav et al [32]. D1 regularization layer randomly drop units from the DCNN along with their weights during training time. This has the effect of training an ensemble of neural networks were 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 un-thinned original network with smaller weights. The last two layers are fully connected layers. FC8 is a traditional multilevel perceptron layer. In fully connected layers every neuron in the previous layer is connected to every neuron of next layer. The output from convolution and sub-sampling layers extract deep features of the input image and fully connected layers use these features for classifying input image. Output layer is a binary classification layer with single sigmoid neuron unit who's on-off state provide class information of input NSLKDD image instance.

3.2 Hyper-Parameter Selection

In process of deep learning, hyper parameters include such 'higher-level' properties of the model which cannot be learned from training set but have profound impact on learning capacity and accuracy of the model. Some hyper-parameters include learning rate of model, non-linearity, choice of objective function, regularization, parameter update method (optimizer), initial weight initializations, mini-batch size of input and number of training epochs to name a few. A learning algorithm $\mathcal{A}$ maps X^{train} from G_x to f through optimization of hyper-parameters λ. The problem of determining good values for λ is called Hyper-parameter optimization.

$$\lambda^{(*)} = argmin_{\lambda \in \Lambda} \mathcal{E} \sim G_x \, [\mathcal{L}(x; \mathcal{A}_\lambda(X^{train}))\,]$$

In general it is difficult to perform optimization implied by above-mentioned equation. With respect to expectation over G_x, the technique of replacing expectation with mean over validation set X^{valid} whose elements are drawn I.I.Ds , where $x \sim G_x$ is used, is called cross-validation. The hyper-parameter problem in general is addressed by following equations.

$$\lambda^{(*)} = argmin_{\lambda \in \Lambda} mean_{x \in X^{valid}} \left[\mathcal{L}\left(x; \mathcal{A}_\lambda(X^{train})\right)\right]$$
$$= argmin_{\lambda \in \Lambda} \Psi_\lambda$$
$$= argmin_{\lambda \in \lambda^{(1)},\ldots,\lambda^{(S)}} \approx \hat{\lambda}$$

Knowing little about hyper-parameter response surface Ψ or search space Λ, the dominant strategy is to choose some number (S) of trial points $\lambda^{(1)}, \ldots, \lambda^{(S)}$, to evaluate $\Psi(\lambda)$ for each one, and return the Hyper-parameter set λ that produced best results as $\hat{\lambda}$. Commonly used strategies for hyper-parameter optimization are grid-search and manual search, but their effectiveness is limited. Grid search forms the set of trials by assembling every possible combination of hyper-parameter values which results in exponential increase in trials and require enormous computing resources. Manual search, on the other hand, suffers the draw-back of difficulty in reproducing results.

Bergstra and Bengio [33] purposed Random-Search strategy for choosing trial-set $(\lambda^{(1)}, \ldots, \lambda^{(S)})$. As the uncertainty arising from X^{valid} being a sample from G_x makes the test-set score of best model among $\lambda^{(1)}, \ldots, \lambda^{(S)}$ a random variable z, this z is modeled by a *Gaussian* mixture model whose S components have means $\mu_s = \Psi(test)(\lambda(s))$, variance $\sigma_s^2 = \mathcal{V}(test)(\lambda(s))$ and weights W_s. The performance z of best model in an experiment of S trials has mean μ_z and standard error σ_z^2 given by following equations:

$$\mu_z = \Sigma_{s=1}^{S} \omega_s \mu_s \text{ and}$$

$$\sigma_z^2 = \Sigma_{s=1}^{S} \omega_s \, (\mu_s^2 + \sigma_s^2) - \mu_z^2$$

The weights ω_s can be estimated by drawing validation scores Z(s) from general normal distribution with means $\Psi(test)(\lambda(s))$ and variance $\mathcal{V}(test)(\lambda(s))$ and counting how often trial generates a winning score. According to [33] validation scores are typically relatively close and few tens of hypothetical draws are sufficient. As compared to *Grid-Search*, *Random-search* strategy make efficient use of limited computational budget to find better model parameters by performing effective search over otherwise large configuration space. We employed Random-search strategy for hyper-parameter optimization and implementation details and results of hyper-parameter tuning are provided in implementation section.

4. Implementation

This section describes the experiment setup, implementation details and hyper-parameters fine-tuning process of the model described in **Fig. 1**.

4.1 Experimental Setup

Hardware setup used for implementing proposed model included:

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

Software toolchain used to implement the model consist of IPython development environment using Keras 2.0 on Theano [34] backend and nVidia cuda 8.0 [35] Training and testing data is manipulated in form of numpy arrays. Python Sci-kit learn library is used to implement other classifiers for comparisons.

4.2 Hyper-Parameter Optimization

For Hyper-parameter optimization of proposed IDS, randomized search approach devised by Bergstra et al. [33] is employed. Randomized search approach efficiently uses limited computational budget to find better model parameters by performing effective search over otherwise large configuration space. **Table 4** provides top 5 models and their hyper-parameters along with accuracy on Training Dataset selected by randomized search algorithm.

Table 4. Hyper-parameters of Top five models selected by RandomizedSearch Algorithm

Sr #	Activation	Kernel_init	Bias	Optimizer	Batch	Epoch	Learn rate	Loss Function	Train Accuracy
1	Softsign	He_normal	Yes	Adadelta	64	15	0.1	L2 Loss	0.9845
2	softsign	glorot_normal	Yes	Adadelta	128	10	0.1	Binary Crossentropy	0.9823
3	softsign	Lecun_uniform	Yes	Adam	256	15	0.01	L2 Loss	0.9766
4	relu	glorot_uniform	No	Adamax	128	10	0.01	L2 Loss	0.9712
5	tanh	glorot_normal	Yes	Adam	128	10	0.05	MSE	0.9626

The selected hyper-parameters include *softsign* activation, He_normal kernel initialization, Adadelta optimizer with batch size of 64 instances. Although learning rate of 0.1 was part of configuration space, Adadelta does not require learning rate. Additional hyper-parameters of proposed model included output layer of single sigmoid unit, drop-out rate of 0.5 and zero-padding at each convolution layer input. A brief introduction of selected hyper-parameters is as follows:

Softsign.

Softsign is a non-linearity which is considered an alternative to *tanh* because of its resistance to saturation as compared to hard-clipped functions because of its smoother asymptotes. Softsign is represented by following equation:

$$softsign\,(x) = \frac{x}{1 + |x|}$$

He_Normal.

He_normal is a kernel initialization scheme proposed by [36] which is built on the work of [37], and allows for faster convergence of deeper CNNs. In forward propagation case, the central idea of He_normal is to investigate the variance of the responses in each layer and design initialization in a way that should avoid reducing or magnifying the magnitudes of input signals exponentially. This is achieved by drawing initializations from a zero-mean normal distribution whose standard distribution is $\sqrt{\frac{2}{n_l}}$. For L layers put together, the initialization design must assure that variance of response of Lth layer conforms to following equation:

$$Var[yL] = Var[y1].\left(\Pi_{l=2}^{L}\frac{1}{2}n_1 Var[w_l]\right)$$

The above product is expected to take a proper scalar (e.g. 1). As described by [36], a sufficient condition is given as follows:

$$\frac{1}{2}n_l\,Var[w_l] = 1\ \forall l \in L$$

Fulfilling abovementioned condition enables kernel initializations which provide many desirable properties for training deep CNNs.

Adadelta.
Adadelta is parameter update mechanism proposed by [38] which provides per-dimension learning rate method for gradient descent. According to [38], Adadelta dynamically adapts over time using only first order information and produces minimal computational overhead without requiring manual tuning of learning rate and shows robustness to noisy gradient information, various data modalities, different model architectures and selection of hyper-parameters. *Adadelta* is represented by following equation:

$$\Delta x_t = -\frac{RMS[\Delta x]_{t-1}}{RMS[g]_t}$$

where Δx_t is parameter update at time t, and $RMS[g]_t$ is exponentially decaying average of RMS at t.

L2 Loss:
L2 loss is an objective/loss function which is minimized by updating weights through back-propagation. It is defined as the mean of absolute squared differences between true labels and predicted labels of classifier. L2 Loss is computed by taking the average of all squared differences as shown below.

$$L(w) = E||y - \bar{y}||^2$$

where y represents actual labels, $\bar{y}$ represents predictions of classifier and E represents expectation. L2 Loss has the intuitive interpretation of heavily penalizing peaky weight vectors and preferring diffuse weight vectors, hence encouraging the network to use all of its inputs a little rather that some of its inputs a lot.

4.3 Classifier implementations for comparison

For comparisons, we used Sci-kit learn [39] implementations of eleven Binary classifiers and trained them on un-raveld version of Training Datasets. These classifiers included Extreme Learning Machine [10] with three different hidden layers namely MLP layer, RBF layer and Generalized layer [40], RBF SVM, k-NN with 08 neighbors, Decision Tree (J48) with 5 node depth, Naïve Bayes, Random-Forest with 10 J48 estimators, Quadratic Discriminant Analysis and Mulitlevel perceptron classifiers. Important hyper-parameters for these shallow models are presented in **Table 5**.

5. Experimental Results and Evaluations

This section presents the results of the model and relevant comparisons with state of art. As mentioned earlier, to the best of our knowledge, DCNNs have not been used for intrusion detection problem so we implemented and trained other classifiers mentioned in previous section with same training and testing Dataset. Performance comparisons with IDS proposed in literatures are also provided.
For classification, accuracy is considered a harsh metric so other prominent metrics for quality of binary classification including Receiver Operating Characteristic (ROC), Area under Curve (AuC), Precision-Recall Curve and mean Average Precision (mAP) is provided. These

evaluation metrics are calculated using confusion matrix which presents four measures as follows:

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

Accuracy. Accuracy is defined as ratio of the number of correctly classified anomalous and normal instances to total number of all instances.

$$Accuracy = \frac{TP + TN}{(NormalCount + AnomalousCount)}$$

True positive Rate (TPR): TPR is also called sensitivity or recall and it is defined as TP/ (TP+FN). This metric corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

False Positive Rate (FPR): FPR is defined as FP/ (FP+TN). This corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points. Both FPR and TPR are used to calculate ROC curve and Area under ROC curve.

Results of all previously mentioned classifiers are taken on both NSLKDDTest+ and NSLKDDTest21 datasets and comparison is provided thereof.

Table 5. Important parameters for other Binary classifiers used for comparison

Classifier	Hyper-Parameters
Decision Tree	Max Depth=5 , Split Quality Measure = 'gini', Max features considered for each best split = 8
Random Forest	Max Depth =5, No. of Estimators = 10, Split Quality Measure = 'gini', Max features considered for each best split = 5
QDA	Priors=None, Regularization Parameter = 0.01, Rank Estimation Threshold = 0.0001
Multilevel Perceptron	Hidden layer Units = 100, Activation = relu, Solver = Adam, L2 Penalty = 0.01, Learning rate = 0.001, epochs = 200
Nearest Neighbor	Neighbors = 8, Algorithm = Ball Tree, Leaf size = 30, Distance Metric = Minkowiski
RBF SVM	Kernel = RBF, Gamma = 1/41, epochs = 2000, Length scale =1, Length scale bounds = (1e-5, 1e5)
Naïve Bays	Default scikit-learn parameters
ELM MLP	Hidden Layer Units = 512, activation=Tanh, Hidden Layer= MLP, epochs = 100
ELM RBF	Hidden Layer Units = 512, activation=Tanh, Hidden Layer= RBF, epochs = 100
ELM Generalized	Hidden Layer Units = 512, activation=grbf, epochs = 100 , Creates a random layer of radial basis function units proposed by Navarro et al.[40]

5.1 Receiver Operating Characteristics (ROC) Curve

RoC is a plot of False positive rate (FPR) against True positive rate (TPR) of binary classifiers which 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 classifier. RoC curves of implemented classifiers for both *NSLKDDTest+* and *NSLKDDTest21 dataset* are shown in **Fig. 2** and **Fig. 3** respectively.

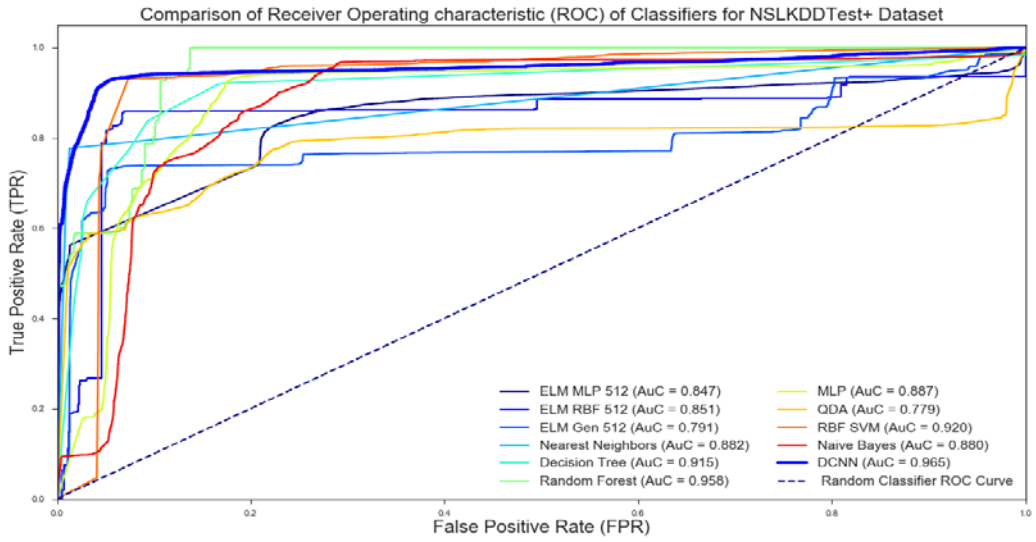

Fig. 2. Comparison of Receiver Operating Characteristics of various classifiers on NSLKDDTest+ Dataset (Top left Bold line shows the results of proposed DCNN based IDS)

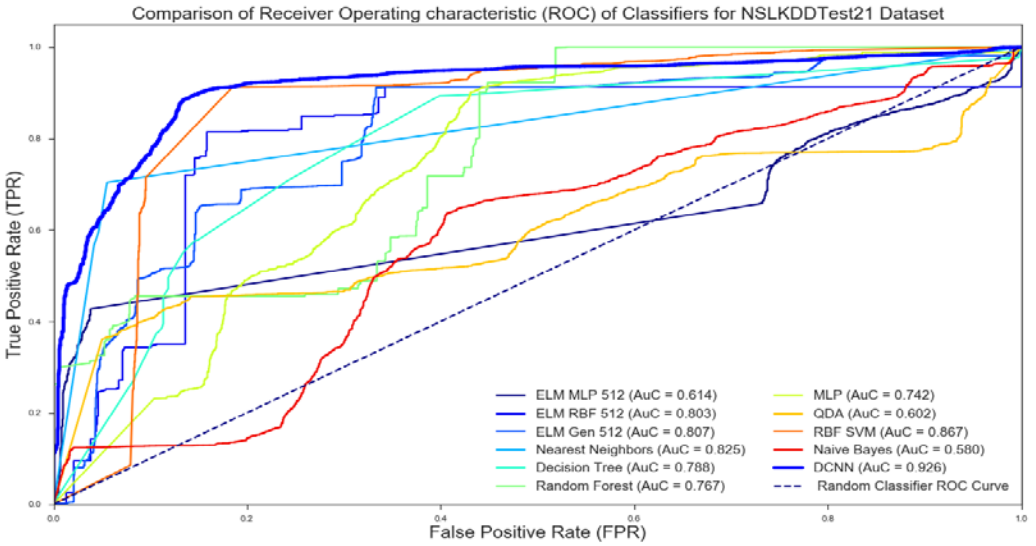

Fig. 3. Comparison of Receiver Operating Characteristics of various classifiers on NSLKDDTest21 Dataset (Top-left Bold line shows the results of proposed DCNN based IDS)

In **Fig. 2**, closest graph to top left border is that of DCNN model followed by Random-Forest and SVM with Radial Basis kernel which shows superior performance of proposed approach in comparison with remaining classifiers to make quality predictions.
Proposed DCNN model again outperforms other classifiers in **Fig. 3** which show the RoC curves of algorithms for NSLKDD*Test21* dataset. Best RoC ratio is that of proposed DCNN model followed respectively by SVM with RBFKernel and k-NN classifier.

5.2 Area under ROC Curve (AuC)

Area under RoC Curve (AuC) is a measure of how well a binary classifier can predict labels (i.e Anamolous and Normal traffic). The AuC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive (Anomalous) record higher than a randomly chosen negative (Normal) one. A perfect binary classifier has AuC =1 and any value of AuC < 0.5 shows bad performance of classifier. A larger AuC is usually better. Value greater than 0.80, 0.90 and 0.95 is deemed good, very good and excellent respectively. The AuC values for both NSLKDD*Test+* and NSLKDD*Test21* datasets are shown in legends part of **Fig. 2** and **Fig. 3**. Top 5 AuC scores for both test datasets are shown in **Table 6** which depicts the improvement of DCNN based IDS.

Table 6. Top 5 Area under RoC Curve results among implemented Classifiers for NSLKDD*Test+* and NSLKDD*Test21* Datasets

Classifier Name	AuC for NSLKDDTest+	Classifier Name	AuC for NSLKDD*Test21*
DCNN	**0.965**	**DCNN**	**0.926**
Random-Forest	0.958	RBF SVM	0.867
RBF SVM	0.920	k-NN	0.825
Decision Tree	0.915	ELM Generalized	0.807
MLP	0.887	ELM RBF	0.803

5.3 Accuracy

Accuracy results of implemented classifiers are shown in **Fig. 4**. As shown in figure, DCNN based IDS retains top accuracy for both NSLKDD*Test+* and NSLKDD*Test21* datasets. Proposed DCNN showed accuracy of 85.22% for NSLKDDTest+ and 69.59% for NSLKDDTest21 respectively. 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. This means that trained models have never seen these attacks during training as they were not available in training data.

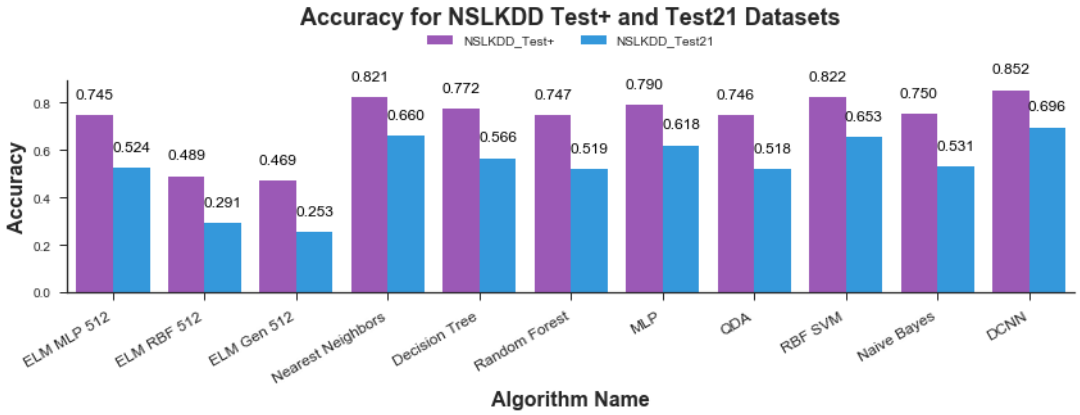

Fig. 4. Comparison of Classification Accuracy of Algorithms for NSLKDD*Test+* and NSLKDD*Test21* Datasets

Comparison of DCNN based IDS with Accuracy results reported in literature is shown in **Table 7** while **Table 8** shows the hyper parameters of DNN based models of Intrusion Detection from Lieterature.

Table 7. Comparison with reported accuracies from literature

Classifier	*Test+* (%)	*Test21* (%)	Classifier	*Test+* (%)	*Test21* (%)
DCNN(Proposed)	**85.22**	**69.56**	Random Tree [16]	82.02	66.16
J48 [16]	81.05	63.97			
Naïve Bays [16]	76.56	55.77	NNRw1 [22]	82.41	67.06
Random Forest[16]	80.67	63.25	NNRw2 [22]	84.12	68.82
MLP [16]	77.41	57.34	Deep Autoencoder (AE) [25]	83.34	Not Reported
SVM [16]	69.52	42.29	Denoising AE [41]	88.65	Not Reported

Table 8. Hyper-Parameters of DNN based models from Literature

Model	Parameters
NNRw1[22]	Fuzziness methodology, Uniform distribution for weight and biases, Init Interval= [0, θ]
NNRw2[22]	Clustering for flag and services feature, Fuzziness methodology, Uniform distribution for weight and biases, Init Interval= [0, θ]
Deep Autoencoder[25]	Reduced Boltzmann Machines with 10 Bernaulli-Gaussian unit for bottleneck layer, Layer-wise pre-training, Loss Function = Binary Crossentropy, Epochs 1500, K-NN Classification layer
De-noising Autoencoder [41]	Hidden layer units = 30, Noise Rate = 10%

5.4 Precision-Recall Curve and Mean Average Precision

Precision is a measure of relevancy of results, while recall is a measure of how many truly relevant results are returned. High precision relates to a low false positive rate, and high recall

relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all 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 puzzel piece of classifier performance, they are combined to form Precision-Recall curve which presents relationship between them in more meaningful manner. The relationship between recall and precision can be observed in the stairstep area of Precision-Recall curve - at the edges of these steps a small change in the threshold considerably reduces precision, with only a minor gain in recall.

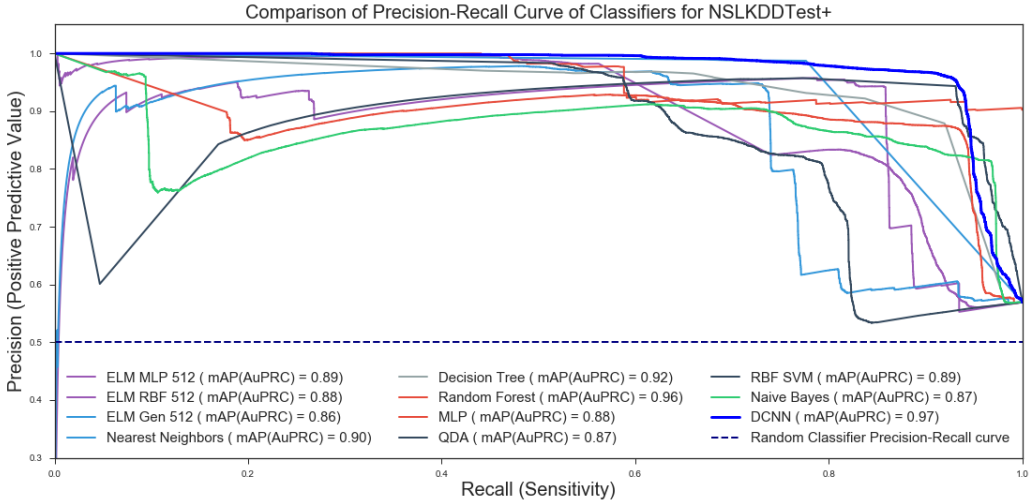

Fig. 5. Precision-Recall Curve of IDS classifiers for NSLKDD*Test+* Dataset (Top-right to left bold line shows the curve for proposed Deep convolution neural network for intrusion detection)

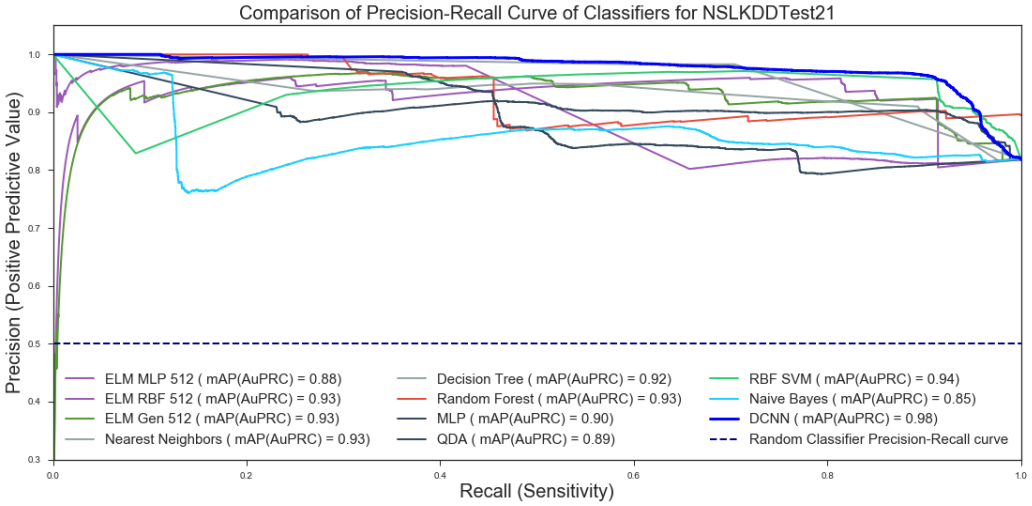

Fig. 6. Precision-Recall Curve of IDS classifiers for NSLKDD*Test21* Dataset (Top-right to left bold line shows the curve for proposed DCNN for IDS)

Perfect precision-recall curve (PRC) shows combination of lines from top-left corner to top-right corner and further down in precision-recall space. This means that a classifier showing abovementioned pattern of PRC maintain a high recall rate as well as high precision rate on different thresoholds. **Fig. 5** shows precision-recall curves of classifiers for

NSLKDD*Test+* dataset and mean Average Precision (mAP) which is shown as area under precision-recall curve in legends section of **Fig. 5**. **Fig. 6** shows the same for NSLKDD*Test21* dataset. As depicted in figures, PRC of proposed DCNN maintained both higher sensitivity and higher precision on different thresholds of the stairstep. This result is also supported by highest scores of Area under PRC achieved by DCNN which are 0.97 and 0.98 for NSLKDD*Test+* and NSLKDD*Test21* respectively.Top 5 mAPs (Area under Precison-Recall curve) are shown in **Table 8**.

Table 9. Top 5 mean Average Precision values (mAP) among implemented Classifiers for NSLKDDTest+ and NSLKDDTest21 Datasets (mAP is also known as Area under Precision-Recall Curve)

Classifier Name	mAP for NSLKDD*Test+*	Classifier Name	mAP for NSLKDD*Test21*
Proposed DCNN	0.97	Proposed DCNN	0.98
Random Forest	0.96	RBF SVM	0.94
Decision Tree	0.92	ELM RBF	0.93
k-NN	0.90	ELM Gen	0.93
RBF SVM	0.89	Random Forest	0.93

5.4 Timing for Model Training and Evaluation

In this sub-section, we provide the training and testing time of algorithms used in this study.

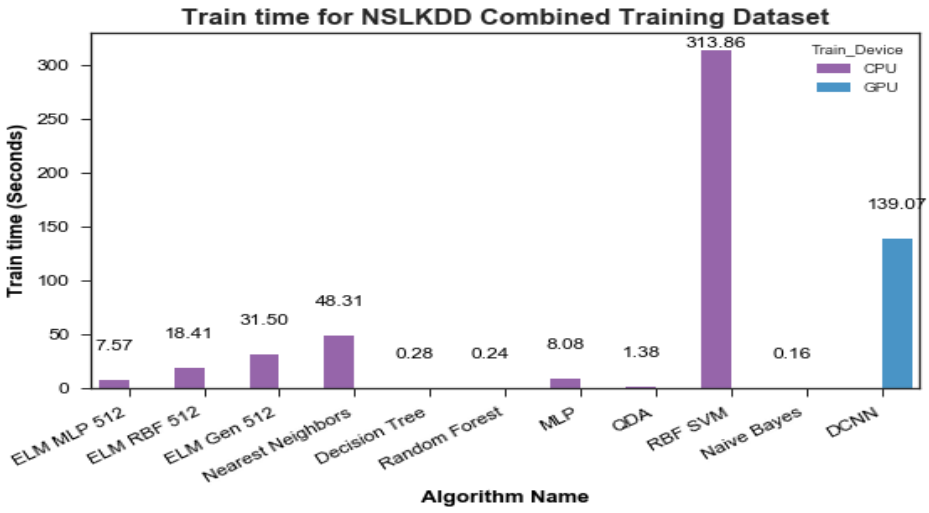

Fig. 7. Training Time in seconds for Different algorithms used in experiments

For DCNN, GPU is used as training and testing device while remaining classification algorithms were trained and tested using CPU. SVM with RBF kernel proved to be the most expensive algorithm for training the model and took approximately 314 seconds. Proposed DCNN model was trained for 20 epochs using GPU in 139 seconds. Remaining algorithms took each under 100 seconds for training. Training times of classification algorithms used in this study are shown in **Fig. 7**.

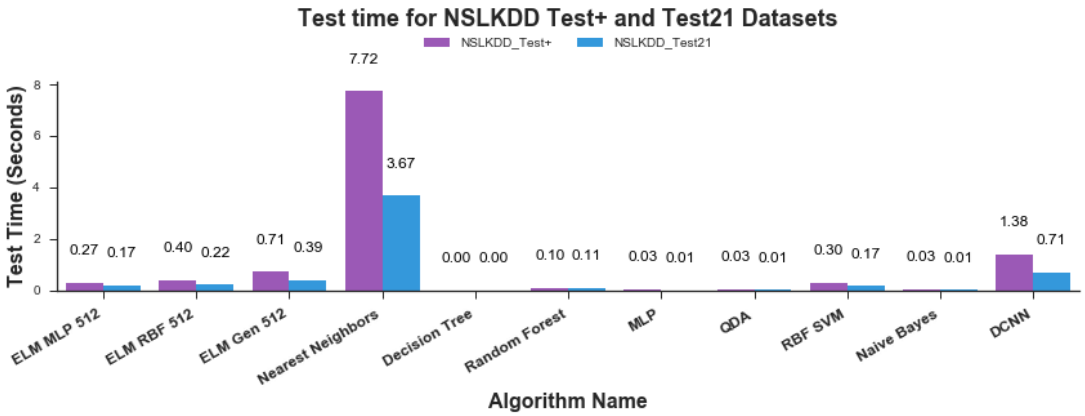

Fig. 8. Test Time for Different algorithms used in experiments for both NSLKDDTest+ and NSLKDDTest21 Datasets

For evaluation of test Datasets, k-NN proved to be the most expensive algorithm and took approximately 8 and 4 seconds for NSLKDD*Test+* and NSLKDD*Test21* datasets respectively. DCNN took 1.39 and 0.71 seconds for evaluating above-mentioned test Datasets. The fastest evaluation was performed by Decision Tree algorithm, which took 5 and 3 milliseconds for evaluating test datasets. Evaluation times for classification algorithms used in this study are shown in **Fig. 8**.

From the results and comparisons shown in figures, it can be inferred that DCNNs show promise as new technology for information security in general and for intrusion detection in particular. Proposed model showed better or comparable results with already established techniques and methods for anomaly detection.

6. Conclusion

In this paper, we proposed, implemented and analyzed a deep convolutional neural network (DCNN) based intrusion detection system (IDS). Proposed DCNN model was trained using GPU on NSLKDD training dataset and evaluation of the same was performed on NSLKDD*Test+* and NSLKDD*Test21* datasets. Performance of DCNN was compared with results from literature and other classification algorithms using well-known metrics including Receiver operating characteristics (RoC) curve, Area under RoC curve (AuRoC), Accuracy, precision-recall-curve and mean Average precision. Proposed model achieved classification accuracy of 85.22 % and 69.56% for NSLKDD*Test+* and NSLKDD*Test21* respectively. Results showed that like other application domains, DCNNs are a promising technology for information security applications. Our future research will be directed towards investigating DCNNs and other deep neural network architectures as feature engineering constructs from raw network data and comparing quality of extracted deep network features with conventional network traffic features for information security applications.

References

[1] D. E. Denning, “An intrusion-detection model,” *IEEE Trans. Softw. Eng.*, no. 2, pp. 222–232, 1987. Article (CrossRef Link)

[2] M. Luo, L. Wang, H. Zhang, and J. Chen, “A Research on Intrusion Detection Based on Unsupervised Clustering and Support Vector Machine,” in *Proc. of 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, pp. 325–336, 2003. Article (CrossRef Link)

[3] X. Zhu and A. B. Goldberg, “Introduction to Semi-Supervised Learning,” *Synth. Lect. Artif. Intell. Mach. Learn.*, vol. 3, no. 1, pp. 1–130, Jan. 2009. Article (CrossRef Link)

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. Article (CrossRef Link)

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

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, pp. 1097–1105, 2012. Article (CrossRef Link)

[7] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis. IJCV*, vol. 115, no. 3, pp. 211–252, 2015. Article (CrossRef Link)

[8] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A Detailed Analysis of the KDD CUP 99 Data Set,” in *Proc. of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, Piscataway, NJ, USA, pp. 53–58, 2009. Article (CrossRef Link)

[9] 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 Explor.*, vol. 2, p. 81, 2000. Article (CrossRef Link)

[10] 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. Article (CrossRef Link)

[11] Y. Liao and V. R. Vemuri, “Use of K-Nearest Neighbor classifier for intrusion detection,” *Comput. Secur.*, vol. 21, no. 5, pp. 439–448, Oct. 2002. Article (CrossRef Link)

[12] S. Mukkamala, G. Janoski, and A. Sung, “Intrusion detection using neural networks and support vector machines,”, pp. 1702–1707, 2002. Article (CrossRef Link)

[13] W. L. Al-Yaseen, Z. A. Othman, and M. Z. A. Nazri, “Multi-level hybrid support vector machine and extreme learning machine based on modified K-means for intrusion detection system,” *Expert Syst. Appl.*, vol. 67, pp. 296–303, Jan. 2017. Article (CrossRef Link)

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

[15] A. A. Ghorbani, W. Lu, and M. Tavallaee, "Network Intrusion Detection and Prevention," *Boston, MA: Springer US*, vol. 47., 2010. Article (CrossRef Link)

[16] M. Tavallaee, “An adaptive hybrid intrusion detection system,” University of New Brunswick, 2011. Article (CrossRef Link)

[17] A. Solanas and A. Martínez-Ballesté, "Advances in artificial intelligence for privacy protection and security," *Hackensack, N.J.: World Scientific*, ISBN: 978-981-4472-03-6, 2010.

[18] P. Laskov, P. Düssel, C. Schäfer, and K. Rieck, “Learning Intrusion Detection: Supervised or Unsupervised?,” in *Proc. of 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, pp. 50–57, 2005. Article (CrossRef Link)

[19] O. E. David and N. S. Netanyahu, “Deepsign: Deep learning for automatic malware signature generation and classification,” in *Proc. of 2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2015. Article (CrossRef Link)

[20] N. Gao, L. Gao, Q. Gao, and H. Wang, “An Intrusion Detection Model Based on Deep Belief Networks,” pp. 247–252, 2014. Article (CrossRef Link)

[21] Z. Wang, "The Applications of Deep Learning on Traffic Identification," *blackhat 2015*, 2015. Article (CrossRef Link)

[22] 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," *Inf. Sci.*, vol. 378, pp. 484–497, Feb. 2017. Article (CrossRef Link)
[23] J. Kim, N. Shin, S. Y. Jo, and S. H. Kim, "Method of intrusion detection using deep neural network," in *Proc. of Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*, pp. 313–316, 2017. Article (CrossRef Link)
[24] K. Alrawashdeh and C. Purdy, "Toward an Online Anomaly Intrusion Detection System Based on Deep Learning," in *Proc. of Machine Learning and Applications (ICMLA), 2016 15th IEEE International Conference on*, pp. 195–200, 2016. Article (CrossRef Link)
[25] M. Yousefi-Azar, V. Varadharajan, L. Hamey, and U. Tupakula, "Autoencoder-based feature learning for cyber security applications," in *Proc. of Neural Networks (IJCNN), 2017 International Joint Conference on*, pp. 3854–3861, 2017. Article (CrossRef Link)
[26] O. Zhang, "Strategies to encode categorical variables with many categories," Feb-2017. Article (CrossRef Link)
[27] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proc. of Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1113–1120, 2009. Article (CrossRef Link)
[28] Statistical Consulting Group, "Contrast Coding Systems for categorical variables," Feb-2011. [Online]. Article (CrossRef Link)
[29] W. Mcginnis, "Beyond One-Hot: an exploration of categorical variables," Jul-2017. Article (CrossRef Link)
[30] W. Mcginnis, "BaseN Encoding and Grid Search in categorical variables," Jul-2017. Article (CrossRef Link)
[31] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. Article (CrossRef Link)
[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014. Article (CrossRef Link)
[33] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *JMLR*, p. 305, 2012. Article (CrossRef Link)
[34] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *ArXiv E-Prints*, vol. abs/1605.02688, May 2016. Article (CrossRef Link)
[35] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable Parallel Programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008. Article (CrossRef Link)
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. of Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015. Article (CrossRef Link)
[37] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. of Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256, 2010. Article (CrossRef Link)
[38] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *CoRR*, vol. abs/1212.5701, 2012. Article (CrossRef Link)
[39] F. Pedregosa *et al.*, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011. Article (CrossRef Link)
[40] F. Fernández-Navarro, C. Hervás-Martínez, J. Sánchez-Monedero, and P. A. Gutiérrez, "MELM-GRBF: A modified version of the extreme learning machine for generalized radial basis function neural networks," *Neurocomputing*, vol. 74, pp. 2502–2510, 2011. Article (CrossRef Link)
[41] R. C. Aygun and A. G. Yavuz, "Network Anomaly Detection with Stochastically Improved Autoencoder Based Models," in *Proc. of Proceedings of the International Conference on Cyber Security and Cloud Computing*, pp. 193–198, 2017. Article (CrossRef Link)

Dr. Yasir Saleem is currently serving as an Associate Professor in University of Engineering and Technology (UET), Lahore, Pakistan. His research interests include Computer Vision, Image Processing, Computer networks, Information/Network Security, DSP, Power Electronics, Simulation and Control system. During his PhD, he did research work for one semester under supervision of Prof. Dr. Zainal Salam in Renewable Energy and Power Electronics Lab, Faculty of Electrical Engineering, UTM, Malaysia. He is an active researcher and currently supervising postgraduate students at MS and PhD levels. He has authored and co-authored journal and conference papers at national and international level in fields of Electrical and Computer Science and Engineering.

Sheraz Naseer is currently working as an Assistant Professor in University of Management and Technology (UMT), Lahore, Pakistan and PhD scholar at University of Engineering & Technology, Lahore. He holds MS in information security along with distinguished professional certifications of information Security including, CISSP, CoBit and ITIL. He has more than 10 years of experience in Information security and IT. His research interests include Cryptography, Data Driven Security, Intrusion Detection, Malware Detection and application of deep neural networks for Information security.