Sādhanā (2020)45:165
https://doi.org/10.1007/s12046-020-01392-4

Efficient deep learning techniques for the detection of phishing websites

M SOMESHA*, ALWYN ROSHAN PAIS, ROUTHU SRINIVASA RAO and VIKRAM SINGH RATHOUR

Information Security Research Lab, National Institute of Technology Karnataka, Surathkal 575025, India
e-mail: somesha.187co004@nitk.edu.in; alwyn@nitk.ac.in; routhsrinivas.cs15fv13@nitk.edu.in; vikram.rathour339@gmail.com

MS received 18 May 2019; revised 10 January 2020; accepted 12 February 2020

Abstract. Phishing is a fraudulent practice and a form of cyber-attack designed and executed with the sole purpose of gathering sensitive information by masquerading the genuine websites. Phishers fool users by replicating the original and genuine contents to reveal personal information such as security number, credit card number, password, etc. There are many anti-phishing techniques such as blacklist- or whitelist-, heuristic-feature- and visual-similarity-based methods proposed as of today. Modern browsers adapt to reduce the chances of users getting trapped into a vicious agenda, but still users fall as prey to phishers and end up revealing their secret information. In a previous work, the authors proposed a machine learning approach based on heuristic features for phishing website detection and achieved an accuracy of 99.5% using 18 features. In this paper, we have proposed novel phishing URL detection models using (a) Deep Neural Network (DNN), (b) Long Short-Term Memory (LSTM) and (c) Convolution Neural Network (CNN) using only 10 features of our earlier work. The proposed technique achieves an accuracy of 99.52% for DNN, 99.57% for LSTM and 99.43% for CNN. The proposed techniques utilize only one third-party service feature, thus making it more robust to failure and increases the speed of phishing detection.

Keywords. Deep neural network (DNN); long short-term memory (LSTM); convolution neural network (CNN); recurrent neural network (RNN); phishing; heuristic technique; deep learning.

1. Introduction

The Internet has brought revolutionary transformations in social networking, communication, banking, marketing and service delivery. The number of users using these Internet facilities is growing at a drastic rate. However, communication technology is growing to meet human needs. However, adversaries are also evolving to disrupt communication. These adversaries steal sensitive information by tricking the user through malware or phishing websites. Phishing is one of the fraudulent ways in the cyber world. The phisher sends a bait as a replica of the real website and waits for the users to fall as prey. The phisher succeeds when a user becomes a victim by trusting the mimicked page. Recent research scientists have given more attention to phishing attacks to avoid damage to innocent Internet users [1]. Survey of such attacks was conducted by many consortia such as NSFOCUS, Anti-Phishing Working Group (APWG), etc.

APWG is a non-profit international consortium, which analyses phishing attacks reported by its members, including security products, service-oriented organizations, law enforcement agencies, government agencies, trade associations, regional international treaties and communications organizations such as BitDefender, Symantec, McAfee, VeriSign, etc. APWG publishes statistical reports on phishing trends across cyberspace periodically (quarterly or half-yearly). According to the latest APWG (2018) [2] report, 263,538 number of phishing attacks were reported with a growth of 46% when compared with the fourth quarter of 2017. There are many ways by which attackers can design phishing attacks. The medium of attack could be e-mails, websites or malware.

In e-mail phishing, a spoofed e-mail is ostensibly sent to the intended users by some trusted companies or organizations. In website phishing, phishers build websites that are replica of real sites and advertise on other website contents or technology giants such as Facebook, twitter, google, etc. Some of the phishing sites also use security indicators such as Hypertext Transfer Protocol Secure (HTTPS) [2], and the green padlock, which makes it difficult for users to differentiate between real and fake sites.

*For correspondence

Published online: 27 June 2020

There are many methods proposed for detecting and preventing phishing. These techniques can be summarized as follows.

- *Listing-based detection*: Most browsers like Chrome, Mozilla, Opera, etc. maintain the list of blocked and permitted Uniform Resource Locators (URLs). The database of blocked URLs is termed as *blacklist* and permitted URLs as a *whitelist*. In *whitelist* databases, even legitimate sites that are not found in the database entry could be the victim and blocked from the browser access. The *blacklist*-based methods follow the opposite of the whitelist. Instead of maintaining a database of legitimate URLs, they keep a database of phishing URLs. This method fails when phishing sites are latest and not even a day old, known as zero-day phishing sites, are encountered. It may be bypassed with slight URL changes. It is mandatory to update the list more frequently, which seems pretty hectic with the amount of rising of phishing attacks.
- *Heuristic-feature-based detection*: This technique is based on features extracted from phishing sites and is used to detect and prevent phishing attacks. However, the limitation is that the heuristic features are not always guaranteed to exist in all the phishing sites, which may lead to reduced detection rates. Also, this technique could be easily bypassed if appropriate algorithms or detection features are known in advance.
- *Visual-similarity-based detection*: Attackers mimic the target websites by the use of favicons, screenshots, background images and logos such that a user can get tricked easily. There are many techniques [3–6] that use databases of logos, screenshots, favicons and Document Object Models (DOM) of target websites for similarity computation with suspicious sites. If the similarity score is higher than a certain threshold, it implies that the suspicious site has mimicked some legitimate sites, and such websites are declared as phishers. The phishers could easily bypass this security system with a slight change in visual elements without changing its contents.
- *Conventional machine-learning-based detection*: One of the main problems suffered by heuristic detection is that it is not flexible enough to accommodate phishing site changes. Even minor changes could cause such detection to bypass. Therefore, the heuristic model has given flexibility using machine learning techniques to accommodate the changes. In this technique, datasets are prepared to train the machine learning model, and the dataset represents the values of features extracted using a heuristic approach. Some of the algorithms used are Support Vector Machine Decision Tree (SVM-DT), Random Forest (RF), Sequential Minimum Optimization (SMO), Principal Component Analysis Random Forest, J48 tree, Multilayer Perceptron, etc. These algorithms can detect even zero-day phishing attacks when trained with heuristic model features. If training data is vast, these algorithms are even better as they learn most of the possible variations that phishing sites may have. Rao and Pais [1] achieved an accuracy of about 99.5% in detecting phishing sites using machine learning techniques. Also, according to the recent survey [7], the detection of phishing sites with an accuracy of more than 99% could be achieved using machine learning techniques. The performance of the machine learning algorithm depends on the size of the training data, the quality of the extracted features and the values of certain hyperparameters used to optimize the accuracy.
- *Deep-learning-based detection*: Deep learning is a machine learning technique that learns features directly from data. The data may be images, text or sound. Deep learning requires a large amount of labelled data and makes it possible for the Graphical Processing Unit (GPU) to train deep networks in less time. New trends have been made to exploit Deep Neural Network (DNN) techniques such as multi-layer feed-forward network [8], Convolutional Neural Network (CNN) [9] and Recurrent Neural Network (RNN) [10] to detect and prevent phishing attacks. These networks are trained through multi-featured datasets obtained using heuristic methods. Bahnsen *et al* [10] trained the RNN over the URL character sequence. They argued that each character sequence has correlations, i.e. nearby characters in the URL are likely to be connected. These sequential patterns are important because they can be used to improve predictor performance. Le *et al* [9] used CNN to learn sequential URL behaviour. They adopted two techniques that are CNN character level and CNN word level, which identify unique characters and words. Each character or word is represented as a vector and trains the vectors over CNN to learn the sequential behaviour of the URL to identify the phishing URLs.

The robustness of the machine learning algorithms, trained over datasets consisting of values of heuristic methods, has led to the proposal of many methods for dealing with phishing sites. Many works [7, 11–13] use third-party websites such as Google or Bing results, Alexa[1] ranking and WHOIS[2] to detect phishing. However, some of the phishing websites hosted on the compromised domain even bypass such techniques. According to APWG's [14] report, most phishing sites may not last for a day, but phishing sites hosted on a compromised site may live for more than a day. The same statistics refer to the limitations of the existing technique leading to such a drastic increase in phishing over the years. Therefore, there must be a mechanism to prevent phishing attacks with higher accuracy and

[1]https://www.alexa.com/topsites.

[2]https://www.whois.com.

minimize the use of third-party services with minimal features. The heuristic method captures specific and compelling features that are sufficiently robust to detect even zero-day phishing. These methods were used to extract the required features for the training of our multi-layer DNN, Long Short-Term Memory (LSTM) Network and CNN. We also tried to optimize the hyperparameters of these networks to obtain the best possible accuracy with minimal features.

The previous work [1] used RF and their variations as classifiers with a rich feature set for the classification of phishing sites. In our current work, we have used deep learning algorithms such as CNN, LSTM and DNN for the detection of phishing websites. Also, we have used an information gain (IG) algorithm to select the best performing features among our proposed features and used it for classifying the phishing websites. The feature selection resulted in a reduction of features from 18 to 10 with lower dependence on third-party services while achieving the same accuracy as that of the previous work. Out of these 10 features, 6 are existing features proposed by others, and the 4 features are proposed in the earlier work.

Our paper makes the following research contributions:

1. We have proposed the IG algorithm to select the best performing features for phishing URL detection.
2. We have proposed novel DNN-, LSTM- and CNN-based models for phishing URL detection with only 10 features.
3. These proposed models achieved a promising accuracy of 99.52%, 99.57% and 99.43% for DNN, LSTM and CNN, respectively.

The rest of the paper is organized as follows. In section 2, we discuss related work carried out by different researchers using different techniques and algorithms. Section 3 explains the architecture of the proposed work. Section 4 deals with the implementation of the proposed model with used tools and datasets. In section 5, we discuss and capture the results of individual methods with their efficiency and accuracy by incorporating different test levels of the model. In section 6, we list out the limitations of our work, and finally, we conclude our paper in section 7.

2. Related work

The proposed model fits into deep-learning-based method in which less work has been done, and it has been inspired by existing list-based (whitelist [15], blacklist [16, 17]), heuristic [18–22] and machine learning methods [23–27]. These methods were discussed in section 1, and also, in the previous work [1]. In this section, we discuss some of the latest works on deep-learning- and machine-learning-based techniques, which are given as follows:

Marchal *et al* [28] proposed a client-side application that extracts features mainly from URL and content of the website resulting in a 210-feature vector. The authors used a Gradient Boosting algorithm for the classification of phishing sites to achieve a significant detection rate. The use of a large feature vector may include significant time for the feature extraction and classification of URLs. Sahingoz *et al* [29] proposed a phishing detection model from URLs using a machine learning approach. The authors applied 7 different classification algorithms on the Natural Language Processing (NLP)-based features for the classification of phishing URLs. The experimental results demonstrated that the RF algorithm with NLP-based features achieved a significant accuracy of 97.98%.

Li *et al* [30] proposed a stacking model combining Gradient Boosting Decision Tree, XGBoost and LightGBM algorithms for detecting the phishing web pages. The authors extracted features from URL and Hypertext Markup Language (HTML) of the suspicious website. The extracted features contain 8 URL and 12 HTML-based features to generate a feature vector. The vector was fed to the stacked model for the classification and achieved an accuracy of 97.30%. Jain & Gupta [31] proposed a client-side technique that uses features from the URL and source code of the suspicious site for the classification. They applied five machine learning algorithms to identify the best classifier suitable for their dataset. RF had outperformed other classifiers with an accuracy of 99.09%.

Yang *et al* [32] proposed a phishing website detection based on multidimensional features driven by deep learning. The authors propose a direct URL approach where character sequence features are extracted from the URLs using a dynamic category decision algorithm, and deep learning is applied for the classification of websites. The features are extracted from URL, webpage code and webpage text features, which are combined into the multidimensional feature set. This feature set is fed to the CNN–LSTM model for the detection of phishing sites and achieves an accuracy of 98.99%. El-Alfy [33] proposed phishing websites based on probabilistic neural networks and clustering K medoids. This framework combined unsupervised and supervised algorithms for training the nodes. K-medoid technology uses feature selection or transformation, and component analysis is used to reduce space dimensionality. The technique achieved 96.79% accuracy by considering 30 features.

Zhang *et al* [24] proposed SMO for the detection and classification of Chinese phishing e-business websites. To evaluate the model, they used 15 unique and some generic domain-specific features. They have used 4 different machine learning algorithms for the classification of phishing sites. Among all 4 algorithms, SMO performed the best in detecting phishing sites with an accuracy of 95.83%. The disadvantage of this approach is that it works better with Chinese websites only. Bahnsen *et al* [10] proposed an RNN for the classification of phishing URLs using LSTM.

The authors compared the traditional RF tree machine learning algorithm to LSTM with 3-fold cross-validation. RF used 14 features for URL statistical and lexical analysis with an accuracy of 93.5%. However, RNN with direct URLs performs better than the RF tree classification algorithm with 98.7% accuracy without requiring intensive labour and time-consuming manual extraction of features.

Le *et al* [9] use a deep learning model to detect phishing URLs. They use the URLNet framework to learn a non-linear URL embedding for malicious URL detection directly from the URL. To learn the URL embedding, URLNet uses CNN specifically to both characters as well as words of the URL string. The proposed method has similar accuracy for word-level and character level and performs much better than other methods. This method may fail if the phishing sites are represented with short URLs (bitly, goo, tiny, etc.) and data URLs.

Zhao *et al* [34] proposed a Gated Recurrent Neural Network model and showed that Gated Recurrent Unit (GRU) outperformed the RF classifier with 21 features and achieved 2.1% better efficiency than RF, i.e., 98.5%. However, here only URLs are used as datasets and need transforming of all characters into vectors to learn hidden patterns. Hence, GRU needs more time to train and requires system architecture to be optimized for better performance. Mohammad *et al* [35] proposed a model to predict phishing sites based on the self-structuring neural networks. They used 17 features extracted from the URL and source code of the website. These features are used to classify websites in artificial neural networks. This model should be regularly retrained with up-to-date training datasets.

Feng *et al* [36] proposed a novel classification model for the detection of the legitimacy of a given website. They used the Monte Carlo algorithm [38] for training the model and the risk minimization principle to avoid over-fitting in the proposed model. They adopted 30 features from the UCI[3] repository and achieved an accuracy of 97.71%.

Yi *et al* [37] proposed a deep learning framework with two types of feature sets, namely original and interaction features. The original features are extracted from the URL analysis, i.e., presence of special characters (@, _, Unicode), count of dots and age of the domain. The interaction features are extracted from the source code of the website, i.e. in-degree, out-degree, frequency of accessing URL and cookie absence. Deep Belief Network (DBN) is applied to the extracted features and achieved an accuracy of 90% true positive rate and 0.6% false positive rate.

The related works with deep learning classifiers are summarized in table 1. The table gives a comparison between the proposed method and all other approaches using six different metrics. These metrics include the detection of phishing sites that replace textual content with an image (Image-based phishing), detection of phishing sites that contain most of the hyperlinks directed towards a common page (Common Page Detection), detection of phishing sites that are hosted in any language (Language independence), detection of phishing sites that consist of maximum number of broken links (Broken links), detection of phishing sites based on different models and the number of features used for classification of phishing sites.

[3]http://archive.ics.uci.edu/ml/index.php.

3. Proposed work

The goal of this work is to detect the status of a given URL using minimal distinctive features with deep learning classifiers. The architecture of the proposed system is shown in figure 1. The architecture comprises feature extraction, feature selection and classification methodologies. A set of webpage URLs are fed as an input into the feature extractor, which extracts required features from three sources (URL obfuscation, hyperlink and third-party-based). The extracted features are further fed to the IG feature ranking algorithm. The outcome of the algorithm helps in selecting the best performing features by a clear investigation in considering the dependences. The best performing 10 features are further trained through different deep learning methodologies to output the status of the URL as legitimate or phishing.

A detailed description of individual models are as follows.

3.1 *Feature extraction*

The features are extracted from three sources:

- URL obfuscation features,
- hyperlink-based features and
- third-party-based features.

These features are extracted using Selenium with Python language, an HTML parser, and BeautifulSoup for parsing the websites. The selection of prominent features from the extracted features is carried out using the IG mechanism. The IG for the features proposed by Rao and Pais [1] is given in table 2.

3.1.1 *URL obfuscation features* They are the characteristics that can be extracted from the URL itself. These features do not involve the inclusion of website content on third-party services. Before defining different URL-based features, we have to understand the typical URL anatomy. A URL is a specific Uniform Resource Identifier (URI) that is used to locate existing resources on the Internet. It is used when a web client requests the server for resources such as HTML, CSS, images, videos or other hypermedia. A URL usually consists of four or five components. The typical structure of URL is http://www.reg.signin.nitk.com.pk/secure/login/web/index.php. It consists of the following parts.

Table 1. Summary of related work in comparison with proposed work.

Techniques	Image-based phishing	Common page-based phishing	Language independ-ence	Broken links	Models	Features
Rao and Pais [1]	Yes	Yes	Yes	Yes	Machine learning	18
Zhang *et al* [24]	No	No	No	No	SMO	15
El-Alfy [33]	Yes	Yes	Yes	No	PNN *K*-medoid clustering	30
Zhao *et al* [34]	No	No	Yes	No	Gated Recurrent Neural Network	Direct URLs
Mohammad *et al* [35]	No	No	Yes	No	Neural Network	17
Le *et al* [9]	No	No	Yes	No	Convolution Neural Network	Direct URLs
Bahnsen *et al* [10]	No	No	Yes	No	Recurrent Neural Network	Direct URLs
Yang *et al* [32]	No	No	Yes	No	CNN–LSTM Hybrid Network	Direct URLs
Feng *et al* [36]	Yes	Yes	Yes	No	Neural Network	30
Yi *et al* [37]	Yes	No	Yes	Yes	Deep Learning DBN	8
Proposed Model-I	Yes	Yes	Yes	Yes	Deep Learning DNN	10
Proposed Model-II	Yes	Yes	Yes	Yes	Deep Learning LSTM	10
Proposed Model-III	Yes	Yes	Yes	Yes	Deep Learning CNN	10

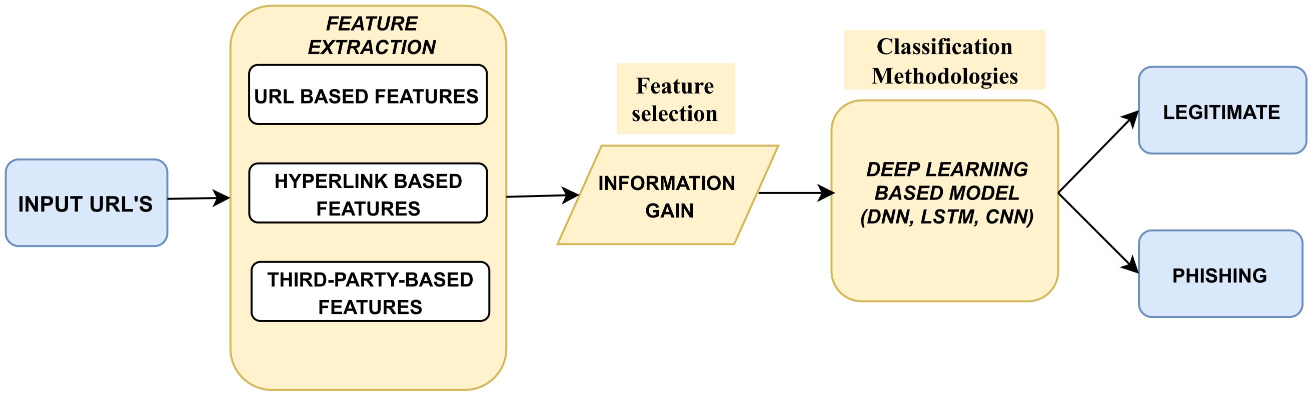

Figure 1. Architecture of proposed model.

Table 2. Information gain of individual features.

Features from Rao and Pais [1]	Information gain
UF1 – Dots in Hostname	0.0874
UF2 – URL with @ symbol	0.00797
UF3 – Length of URL	0.28293
UF4 – Presence of IP	0.00523
UF5 – Presence of HTTPS	0.07321
TF1 – Age of domain	0.29139
TF2 – Page rank	0.88344
TF31 – Website in search engine results-title	0.15664
TF32 – Website in search engine results-copyright	0.16603
TF33 – Website in search engine results-description	0.27909
HF1 – Frequency of domain in anchor links	0.21588
HF2 – Frequency of domain in CSS links, image links and script links	0.04654
HF3 – Common page detection ratio in website	0.40058
HF4 – Common page detection ratio in footer	0.29128
HF5 – Null links ratio in website	0.25015
HF6 – Null links ratio in footer	0.08162
HF7 – Presence of anchor links in website	0.14237
HF8 – Broken links ratio	0.20216

- *Scheme*: The scheme is used to identify the used protocols, Hypertext Transfer Protocol (HTTP) or HTTP with Secure Sockets Layer (HTTPS).
- *Hostname*: The hostname identifies the machine that contains resources. The hostname includes the Generic top-level domain (gTLD) and Country-code top-level domain (ccTLD). In the given example, *reg.signin* indicates subdomain, *nitk* is the primary domain, *com* is gTLD and *pk* is ccTLD.
- *Path*: The path identifies the basic or required information in the host that the web client wants to access. From the given URL example, the path-name is *secure/login/web/index.php*.
- *A query string*: When a query string is used, the path component follows and provides a string of information that the resource can use for some purpose. The query string is usually the name and value pair string. An ampersand separates name and value pairs (&). For example, in the URL http://www.google.co.uk/search?q=url&ie=utf-8, $?q = url\&ie = utf - 8$ is the query string with name and value pairs, respectively, as $q = url$ and $ie = utf - 8$.

In the earlier work [1], five URL obfuscation features (UF1, UF2, UF3, UF4, UF5) were proposed. Out of those five, two features are poorly performing when applied to the IG algorithm as shown in table 2. The best performing three features have been selected, and these features are as follows.

1. UF1: dots in hostname.
2. UF3: lengthy URL.
3. UF5: presence of HTTPS.

3.1.2 *Hyperlink-based features*

These features are extracted from the hyperlinks in the source code of a website. The hyperlink is an electronic document element that connects from one source to another. The web source may be an image, program, HTML document or HTML document element. As mentioned earlier, the previous work [1] consists of eight hyperlink-based features that are used for phishing detection. We have selected six best performing features among the eight features based on the results of IG analysis shown in table 2, and the selected features are as follows.

1. HF1: presence of domain in anchor links.
2. HF2: frequency of domain in image links.
3. HF3: common page detection ratio.
4. HF4: common page detection ratio in footer.
5. HF7: presence of anchor links in HTML body.
6. HF8: broken links ratio.

It may be observed that HF5 and HF6 have performed better in IG, but they have been eliminated since HF3 and HF4 capture these features characteristics [22].

3.1.3 *Third-party-based feature*

In this section, we use third-party services such as WHOIS, Alexa and Search engine for the extraction of third-party-based features. Surprisingly, out of these three features, the Alexa rank-based feature performed significantly better in the IG. Even though other third-party features performed better than the other (URL obfuscation, hyperlink-based) features, we have not considered them in our feature selection to reduce the dependence on third-party services.

TF2: Alexa ranking is a third-party-based service used to classify the phishing sites. The rationale behind this feature is that phishing sites are low ranked, and target websites are highly ranked. This feature checks the rank of a suspicious website in the Alexa database. To calculate the rank, an HTTP request is sent to http://data.alexa.com/data?cli=10&url="+domain and an XML parser is used to get the Alexa ranking.

$$Pagerank = \begin{cases} 0 & \textit{if rank is not found} \\ rank & \textit{otherwise} \end{cases}$$

The selected features from these three sources are highlighted and marked as selected features in table 3.

3.2 *Feature selection*

We have used IG as a ranking criterion to score the features, and by applying the threshold, we have filtered out prominent features. The intuition behind ranking is to evaluate the relevance of the features for the detection of phishing websites. The relevance of feature implies that each feature may be mutually exclusive to each other, but it must not be completely independent of class labels. There must exist a relation between feature and class labels. The

Table 3. Selected features.

Features from Rao and Pais [1]	Selected features (✓)
UF1	✓
UF2	–
UF3	✓
UF4	–
UF5	✓
TF1	–
TF2	✓
TF31	–
TF32	–
TF33	–
HF1	✓
HF2	✓
HF3	✓
HF4	✓
HF5	–
HF6	–
HF7	✓
HF8	✓

features that are irrelevant and have no relation or abysmal role can be discarded. Hence, our primary aim behind using this technique is to rank features based on their relevance and influence on the class labels. This could be used in the feature reduction process. IG [39] is measured based on the entropy of a system. The entropy is defined as a degree of disorder and impurity in the system. IG is defined as a reduction in the impurity, bringing more certainty in the system. For feature ranking purposes, we have calculated IG on the entire dataset. Summarizing, IG looks at each feature in isolation and is calculated on each feature independently. By computing IG of each feature independently, we get a quantitative measure of significance and relevance of this feature on class labels. Computation of information for a feature involves two steps.

1. Compute entropy of the class label for the entire dataset. It can be computed by the formula

$$info(\mathrm{D}) = \sum_{i=1}^{m} p_i \times \log_2 p_i \tag{1}$$

   where $m = 2$, i.e. the total unique number of class labels (phishing, legitimate), and in our dataset, it is two. D represents a feature of a dataset. Hence, each feature has some instances belonging to one class and remaining to another class; p_i represents the probability of instances of D that belong to i^{th} class. We can compute probability p_i by counting the number of instances of D that belongs to i^{th} class and then dividing by the total number of instances of D. Once we get p_i for all i, we use Eq. (1) to calculate the entropy of D.
2. Computation of conditional entropy for each unique value of that feature: The calculation of conditional entropy requires a frequency count of the class label by feature value. The feature value can be continuous as well as discrete.

   i. For discrete-valued features, it can be calculated by the formula:

$$info_A(D) = \sum_{i=1}^{v} \mid \mathrm{D}_i \mid / \mid \mathrm{D} \mid * info(\mathrm{D}_i) \tag{2}$$

   where v is equal to total unique discrete values present in the feature value, D_i represents a count of i^{th} type of value in feature and D is the total count of feature value.

   ii. For continuous-valued features, we have sorted feature value and have divided them in $\sqrt{n}$ bins, where n is equal to the total count of feature values. Now we have n different classes, and it can be treated as discrete-valued features, and Eq. (2) is used to calculate the conditional entropy of continuous features.

Now the information gain (IG) is calculated using the following equation:

$$IG(A) = info(D) - info_A(D). \tag{3}$$

The IG for the features proposed by Rao and Pais [1] is given in table 2.

The classification methodologies have been discussed in detail in section 4.3 along with the formal description of DNN, LSTM and CNN.

4. Implementation

Given a list of website URLs, we have trained and cross-validated a proposed deep-learning-based model to identify as legitimate URL or phishing URL. We have used the Selenium library in Python and Firefox web driver to get screenshots of website URLs, and also to download the source code. We used Beautiful Soup in Python to parse the source code to extract the required features. Screenshots and status codes are further used to verify that contents have not changed while extracting features from source code. Extracted datasets are further examined manually for legitimate URLs; duplicates and unwanted URLs (neither phishing nor legitimate) are removed from the PhishTank dataset. This process is to avoid legitimate sites being treated as phishing and reduce the processing time by avoiding unwanted comparisons.

4.1 *Tools used*

We have implemented Python scripts to extract all features using Python 3.6 from URL and URL content. We collected phishing URLs from the PhishTank[4] website, and legitimate sites from the Alexa databases. When these URLs are fed as inputs to the Python script, all the essential features are extracted and stored in text files. These extracted features are transferred to deep learning algorithms to train and cross-validate so that it can start classifying URLs into legitimate and phishing sites. We have implemented a deep learning algorithm with a TensorFlow package, an open-source machine learning framework implemented on top of Python, which supports parallel computing.

4.2 *Datasets used*

We have used the dataset of Rao and Pais [1] for all our experiments in this paper. The dataset consists of 3526 instances out of which 2119 are phishing sites collected from PhishTank and 1407 legitimate sites collected from the Alexa database. They were further divided into categories of training sets and testing sets of 75% and 25%, respectively, for model evaluation.

[4]http://www.phishtank.com/index.php.

4.3 *Deep learning algorithms*

To evaluate the performance of the feature set, the feature set has been trained and cross-validated against many different parameter combinations. In the multi-feed-forward network, we must gather data based on feature sets and then tune the parameters to achieve maximum accuracy in phishing site classification. It is an essential process in which training networks must set parameters and validate across appropriate values. After attaining the right value, phishing sites can easily be classified with the highest probability. We used Python programming language along with the TensorFlow library to implement deep learning algorithms. From various combinations of hidden layers, we found that the DNN with 5 hidden layers achieved the best results. It can be understood that this permits the features we have extracted in the nonlinear, separable and complex functions to be represented most effectively. The proposed deep feed-forward neural network comprises 7 layers, with 5 hidden layers, one input layer and one output layer. All layers were followed and standardized by the Rectified Linear Unit (ReLU) or sigmoid function. The first four layers were followed by the ReLU function and the output layer using the sigmoid function. The rationale behind batch normalization is that it speeds up training by reducing the internal covariate shift and reducing over-fitting. ReLU activation has replaced sigmoidal or tanh activation functions in hidden layers due to its tendency to learn faster than sigmoidal or tanh, avoiding significant delays in the rate of gradient descent convergence after an initial set of iterations.

4.3.1 *Formal description of DNN* The DNN is a type of machine learning technology. It consists of many common neural network layers. It has one input layer, one output layer and at least one hidden layer, as shown in figure 2.

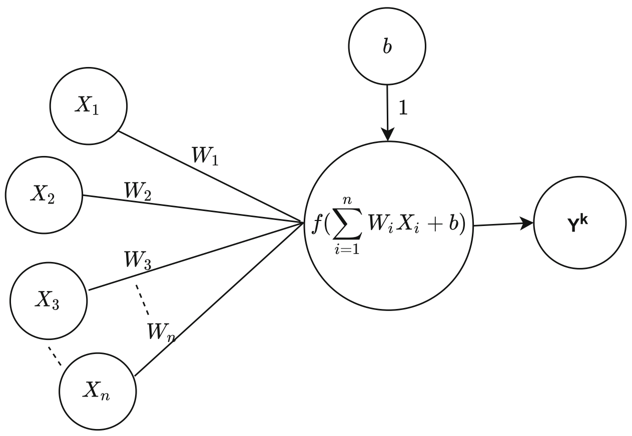

Figure 2. Architecture of simple neuron.

Each layer is composed of the basic computing unit, i.e. the neuron. The neuron is inspired by the biological neuron that performs mathematical functions for the storage of information. This information is transmitted to another neuron, and therefore information propagates in the neural network. A neuron's general mathematical representation is

$$Y^k = \Phi\left(\sum_{k=0}^{k=n} W_{kj}x_j + b_k\right) \tag{4}$$

where Φ is activation function, $W_k \in R^{LB}$ is weight of K^{th} neuron and Y^k is the output of K^{th} neuron. The number of neurons in the input layer depends upon the dimension of datasets or equivalently to the number of features of the dataset, i.e., $X \in R^{LK}$ where L is the total number of the datasets, K is the total number of features in datasets and R represents a real number. The number of neurons in the output layer depends on the number of outputs we want. The number of neurons in the hidden layer is a hyperparameter that needs to be tuned to obtain an optimum result. Since each neuron performs computation, the number of neurons defines the network complexity. Each DNN is a complex mathematical function that adapts itself according to the nature of data. Hence, making the network more complex may result in data over-fitting, i.e. it performs pretty good with training data but fails to achieve good accuracy with unknown data.

Let l = {0,1,2,3,4,5} be the layers in my deep learning model, $Y^{(l-1)}$ be the input to layers {1,2,3,4,5}, $Y^{(l)}$ be output value of layer, where $W^{(l)}$ is weight of layer i that is used for linear transformation of inputs from n layers to output of m layers, $B^{(l)}$ be bias of layer i and $F^{(l)}$ be the associated activation function of each layer. $Y^{(0)}$ is nothing but input layer and $Y^{(l)}$ is output layer.

$$Z^{(l)} = Y^{(l-1)} * W^{(l)} + B^{(l)}, \tag{5}$$

$$Y^{(l)} = F(Z^{(l)}) \tag{6}$$

where * is for matrix multiplication. W values were initialized with Xavier Initialization (the initializer used to initialize random values), and B was initialized with zero. W and B are updated after each iteration in the backpropagation method. Layer 0 is the input layer, layer 6 is output layer 6 and layers 1–5 are hidden layers activated with the ReLU function provided by

$$Y_i^l = \begin{cases} 0 & if \quad Z_i^l \leq 0 \\ Z_i^l & otherwise \end{cases} \tag{7}$$

where i represents i^{th} iteration and l represents l th layer. The intermediate output of our model Y^* is obtained as follows using the sigmoid activation function:

$$Y^* = \frac{1}{1 + \exp -Z^l} \tag{8}$$

where $l = 6$ in case of output layer. The loss function $(L(Y^*, \hat{Y}))$ over entire dataset is defined as sum of cross-entropy between model output and actual output, which is shown as follows:

$$L(Y^*, \hat{Y}) = \sum_{j=1}^{n} [\hat{y_j} \log y_j^* + (1 - \hat{y_j}) \log(1 - y_j^*)] \tag{9}$$

where Y^* is an intermediate output of entire dataset obtained after processing it through deep learning model and $y_j^* \in (0, 1)$ is jth row of Y^* while $\hat{Y}$ is an actual label of our dataset and $\hat{y_j} \in \{0, 1\}$ is jth row of $\hat{Y}$, where 0 represents legitimate site and 1 indicates phishing site. The loss function given earlier is optimized using the Adam Optimizer at every epoch to update parameters and train deep neural model using the backpropagation algorithm. The functional formations represent these features without overfitting, because DNN has 5 hidden layers along with 1 input and 1 output layer.

4.3.2 *Formal description of LSTM* An RNN is a specific type of bio-inspired neural network that can model and learn a sequential data pattern. It learns sequential dependences by learning one sequence at a time, thereby introducing time to neural network modelling.

The RNN has been good at the sequential and time-series dataset, and has proved to be very useful [10, 40]. However, the general recurring network suffers from a vanishing gradient problem or an explosive gradient problem, i.e. it cannot retain memory across a larger path that causes long-term dependences [41, 42]. As a result a long correlation between sequences is not maintained, and the network fails in such circumstances. Hence, LSTM takes care of the long correlation between sequences.

LSTM (figure 3) removes the vanishing gradient problem or exploding gradient problem to avoid long-term dependences. In LSTM, a neuron is replaced by cell memory, which performs the task using activation function for input by forming a linear combination of the dot product of input and weights with bias. LSTM also uses update gate, forget gate and the output gate to avoid long-term dependences.

$$\hat{C}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \tag{10}$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \tag{11}$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f) \tag{12}$$

$$\Gamma_o = \sigma(Wo[a^{<t-1>}, x^{<t>}] + b_o) \tag{13}$$

$$C^{<t>} = \Gamma_u * \hat{C}^{<t>} + \Gamma_f * C^{<t-1>} \tag{14}$$

$$a^{<t>} = \Gamma_o * C^{<t>} \tag{15}$$

Weight matrices W_c, W_u, W_f, W_o, bias vectors b_c, b_u, b_f, b_o, temporary cell state $(\hat{C}^{<t>})$, update gate (Γ_u), forget gate (Γ_f), output gate (Γ_o), cell state $(C^{<t>})$ and activation $(a^{<t>})$ [43] remain the same for all time-steps in a single unit of LSTM network, and are updated after each epoch during the backpropagation method. The long memory is usually called a cell state [Eq. (10)]. The cell state allows the network to store the information coming from the previous cell. It is updated using the value of both the update gate [Eq. (11)] and the forget gate [Eq. (12)]. The forget gates enable the network to forget the information that is not necessary or not relevant by multiplying with 0. It also helps retain information by multiplying by 1. The update gate determines which information should enter the cell memory for storing. The output gate (Eq. (13)) decides which result should move forward to the next hidden layer. For exploring the feasibility of an RNN on our dataset, we have implemented LSTM.

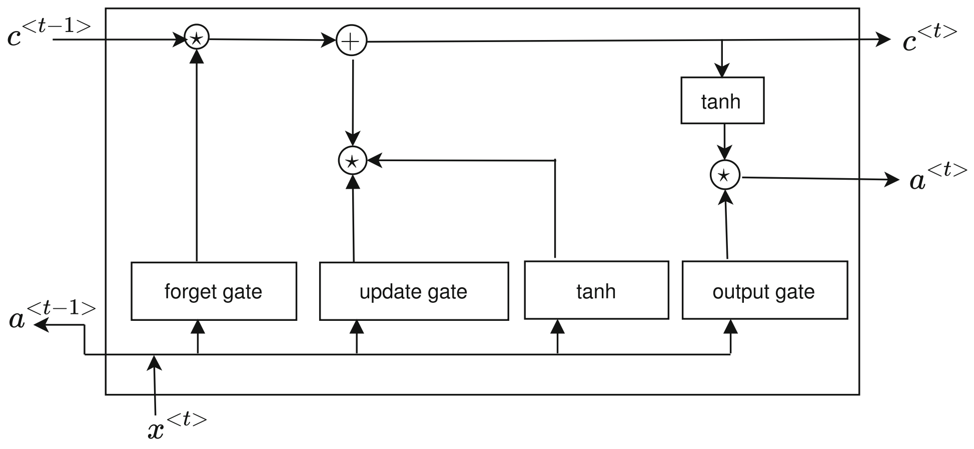

Figure 3. Architecture of LSTM.

In this neural network, we have taken 4 LSTM units performing different mathematical computations that are defined earlier. Each unit has 10 time-steps. Each time-step has 1 output, which is passed as input to the next time-step. The last time-step of the first unit is also passed to the second unit, and so on, and finally, we get output from the last time-step of the fourth unit of LSTM. The obtained output is further densed to a single output and passed to the sigmoid function. The loss function is calculated, and error is optimized using the Adam Optimizer. During back-propagation, each parameter of all 4 units of LSTM is updated. Again the loss function is calculated in each epoch; the network learns when variables are updated. We modified the dataset dimensionality to implement LSTM. We have converted 10 features to 10 time-steps; each time-step consists of 1 feature. Hence, our dataset new dimension is (3526,10,1). Through LSTM, we attempted to find out the possible relationship between different features. Initially, at the first gate, zero vector and first time-step were passed to the first gate. The output from the first LSTM gate passed as input to second, and so on till the tenth gate. The obtained single output from the tenth gate forms a single LSTM unit output, which is again passed as input to the first gate of the second unit. Hence, the output of the previous gate along with the current time-step is fed to the next gate and the output of the previous LSTM unit is fed to the next unit until the fourth LSTM unit. In the fourth unit, its output is densed to 1, which is passed on to the sigmoid activation function (8), and then the loss function (9) is calculated and optimized using the Adam Optimizer.

4.3.3 *Formal description of CNN* CNN is similar to an ordinary DNN. These networks consist of neurons that have weights and biases, which are updated and made to learn. Each of these neurons receives inputs that are converted into a linear combination of dot products of weights and input bias. However, instead of fully connected hidden layers, it performs convolution on input layers $x \in R^{LB}$. Convolution is performed using convolution operator $\otimes$ of length L with stride s, and consists of convolving filter $W \in R^{BK}$.

Generally, CNNs are used with images due to the high correlation between pixels and networks. CNNs can figure out relations and different features using convolutional techniques, which are used in conjunction with the pooling layer, and batch normalization are done before passing it to any activation function [9, 44]. It has also been used in NLP after character encoding due to the correlation between character sequences [9, 45].

In our work, the selected 10 features from the IG algorithm are fed to the CNN model to identify the status of the suspicious site. The proposed CNN model consists of 8 layers (6 convolution and 2 dense layers). In the first layer, the input is passed to the convolution layer and the output of this layer is activated using a tanh function (Eq. (16)). Later the activated output is subjected to batch normalization and pooling. The obtained output is passed to the next convolutional layer. In this way, we have 6 convolutional layers connected sequentially in which the output of one layer is the input of the next. At the seventh layer we densed the output of sixth convolutional layer to 500, again activated using tanh function and then at the end densed it to 1. The output of the tanh function is passed to the sigmoid activation function (8) for output in the range of (0,1). Later the loss function (9) is calculated and is optimized using the Adam Optimizer. Then we have a backpropagation method where variables are updated, and thus network learns.

$$\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1). \tag{16}$$

We have used 10 features extracted in the feature selection process. The size of our dataset is 3526. Hence the dimensionality of our converted dataset is (3526, 10, 1), and it is passed to our CNN model for phishing detection.

5. Results and discussions

We conducted experiments to evaluate the performance of our DNN, LSTM and CNN models with different features and parameters. All experiments were conducted with the same dataset of 3526 instances. Each experiment has been repeated, and data has been randomly selected from the dataset. For evaluating our model, we have used accuracy (Eq. (17)) and error (Eq. (18)) rates as the main evaluation metrics. To calculate them, we considered the phishing sites as condition positive (P), where P represents the total number of phishing sites in our dataset. The legitimate sites are termed as condition negative (N), where N represents the total number of legitimate sites in our dataset. The correctly classified phishing sites are termed as True Positive (TP), which is calculated as the ratio of correctly classified phishing sites to the total number of phishing sites (P). Correctly classified legitimate sites are termed as True Negative (TN), which is calculated as the ratio of correctly identified legitimate sites to the total number of legitimate sites (N).

- *Accuracy (ACC)*: Measures the legitimacy and phishing rate of the total number of websites.

$$\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{P + N}. \tag{17}$$

- *Error Rate (ERR)*: Measures the rate of legitimacy or phishing from incorrectly classified websites.

Table 4. Accuracy of individual features.

Features	Training accuracy (%)	Testing accuracy (%)
UF1	64.94	63.94
UF2	59.90	60.86
UF3	78.03	76.79
UF4	59.90	60.86
UF5	66.94	67.12
TF1	77.34	79.41
TF2	95.83	96.47
TF31	81.96	80.01
TF32	75.57	74.18
TF33	64.78	62.0
HF1	72.04	68.25
HF2	64.48	61.32
HF3	82.37	80.31
HF4	80.33	79.18
HF5	75.04	73.95
HF6	63.39	63.25
HF7	68.75	68.03
HF8	76.96	75.88

$$\mathrm{ERR} = 1 - \frac{\mathrm{TP} + \mathrm{TN}}{P + N}. \qquad (18)$$

5.1 *Validation of selected features using DNN*

Our work of feature retention and rejection based on inferences that we have drawn from IG is listed in table 2. We have retained those features that have higher IG. In this section, we validate our claim of feature selection using DNN.

To validate our selection of features, we have conducted 3 experiments. The first experiment is conducted by supplying all the 18 features of our earlier work [1] to DNN. The overall accuracy obtained using 18 features is 97.95%. The results of the individual accuracy of this experiment are tabulated in table 4. The testing accuracy of individual feature varies from 60.86% (UF2, UF4) to 96.47% (TF2). Based on the accuracy, we have eliminated two URL-based features whose testing accuracy is less than 61% (UF2, UF4). We have eliminated two hyperlink-based features HF5, HF6 since their functionalities are taken care of by HF3 and HF4, respectively. Also, individual accuracy of HF5 and HF6 is less than that of HF3 and HF4, respectively. Experiment 2 is conducted after eliminating 4 features (UF2, UF4, HF5, HF6) from the total set of 18 features. The accuracy chart of training and testing using these 14 features is given in figure 5. The overall accuracy obtained using 14 features is 99.20%. Experiment 3 is conducted to evaluate features by minimizing third-party-based features. The overall accuracy after eliminating four third-party feature is 98.97%.

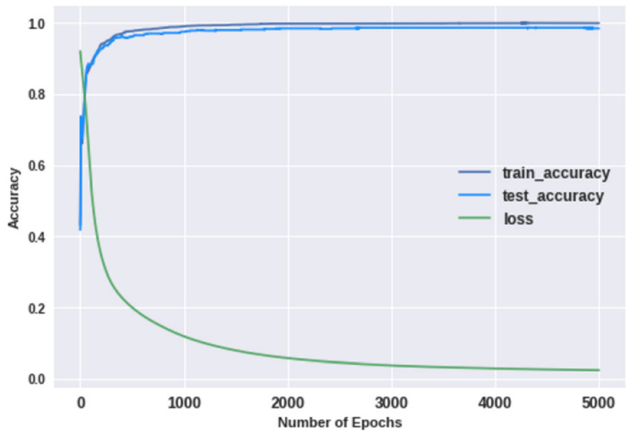

Figure 4. Network performance using 18 features.

The 10 features used in experiment 3 are the same, and the one with the highest IG is given in table 2. The experimental results are in line with the IG, and they validate our selection of features.

Experiment 1 – evaluation of individual heuristic features using DNN: In this experiment the performance of each individual feature has been evaluated, and is given in table 4. This has been done in order to know the individual contribution of each feature in determining accuracy. The features that have higher accuracy in detecting phishing sites have more relevance to class labels. This relevance will be an experimental manifestation of our feature ranking process using the IG algorithm. It will also be an experimental justification of retention and rejection of features using IG. The overall accuracy obtained using 18 features is 97.95%. The accuracy chart using 18 features is shown in figure 4. The obtained accuracy is less than that in our earlier work [1] with the same set of features.

Experiment 2 – evaluation of model using 14 features: In this experiment, we have evaluated our model accuracy using 14 features. The features that we left out are UF2, UF4, HF5, HF6. We have summarized reasons for leaving out these features as follows.

- UF2 and UF4: The first two features have individual accuracy of less than 61%, which is the lowest in table 4. In our IG table 2, two of the lowest values are 0.00797 and 0.00523 for UF2 and UF4, respectively. The lowest accuracy and lowest IG, among other features, validate their rejection in the feature selection process. These two features are the least relevant to class labels, and their individual contributions are the lowest. Hence, our experimental analysis upholds the rejection of these features in order to reduce inhibition offered by the least performing network.
- HF3 vs HF5: HF3 is the ratio of the most common link to the total number of links in the URL web page, and HF5 is the ratio of null links to the total number of

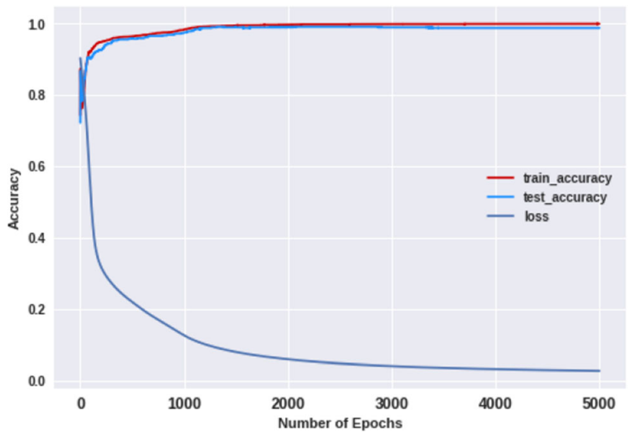

Figure 5. Accuracy chart with 14 features.

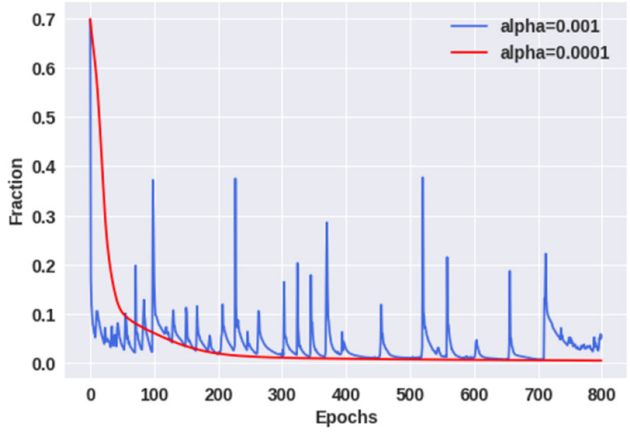

Figure 6. Learning rate with $\alpha = 0.001$ and $\alpha = 0.0001$.

links in the URL web page. If the null link is the most common link, then both are equivalent, and if null links are less or absent, then null link ratio features are of no use. Hence, HF3 contains HF5 [22]. For removing features, we had considered the IG of each feature. In table 4, HF3 individual accuracy is higher than that of HF5. Hence, retention of HF3 is experimentally justified.

- HF4 vs HF6: HF4 is the ratio of the most common link to the total number of links in the footer, and HF6 is the ratio of the null link to the total number of links in the footer. Both will have the same value if the most common link is a null link. If null links are less or absent, it is shallow. HF3, therefore, contains HF5 [22]. In our feature selection process, we have considered higher IG of HF4 over HF6. In our experimental analysis, the individual accuracy of HF4 is higher than that of HF6, which validates our claim.

After removing these features, accuracy has increased to 99.20% and the same can be observed in figure 5.

Experiment 3 – evaluation of features by minimizing third-party-based features: In this experiment, we began by retaining all third-party features (TF1, TF2, TF31, TF32, TF33) shown in table 3. The extraction of third-party features from URL is a time-consuming process. Hence, phishing URL detection cannot be done in a faster time-bound manner. If any of these third-party features are not available then we have to deal with missing data, and the accuracy of the model will drop. Hence we removed all the third-party features, including TF2, which has the highest IG, to test the robustness of our model. The accuracy of our model dropped to 90% after removing all the third-party features. Again we experimented with the inclusion of one third-party feature (TF2) and obtained an accuracy of 98.97% with 5000 epochs.

5.2 *Results with DNN*

Section 5.1 described three experiments using DNN to validate our features by comparing results obtained from the IG ranking algorithm. We have selected 10 best performing features from experiment 3. The individual accuracies of the best 10 features are shown in figure 6. Experiment 4 is conducted using DNN on the dataset of Rao and Pais [1] with selected 10 features. The hyperparameters tuning is performed to optimize the model by selecting the learning rate (α), optimizer, number of hidden layers, number of nodes in layers and number of epochs.

Experiment 4 – evaluation of model by tuning parameters: In experiment 3, we have not fine-tuned the parameters to optimize our model. In experiment 4, fine-tuning of parameters was performed to optimize our DNN model. The parameter fine-tuning process is as follows.

- *Learing rate*: We began with $\alpha = 0.001$, keeping the rest of the features as specified in table 5. At this α value, our model's loss function converged rapidly, as indicated in figure 7 with higher losses. In this case, we got 99.69% training accuracy and 98.97% testing accuracy. Hence we increase α to 0.0001, and the loss function of our model begins to converge, as we can

Table 5. Parameters for DNN.

Layers	Number of units in layers	Learning rate	Optimizer	Epochs	Activation function
6	10, 19, 100, 200, 300, 1	0.0001	Adam Optimizer	6000	ReLU

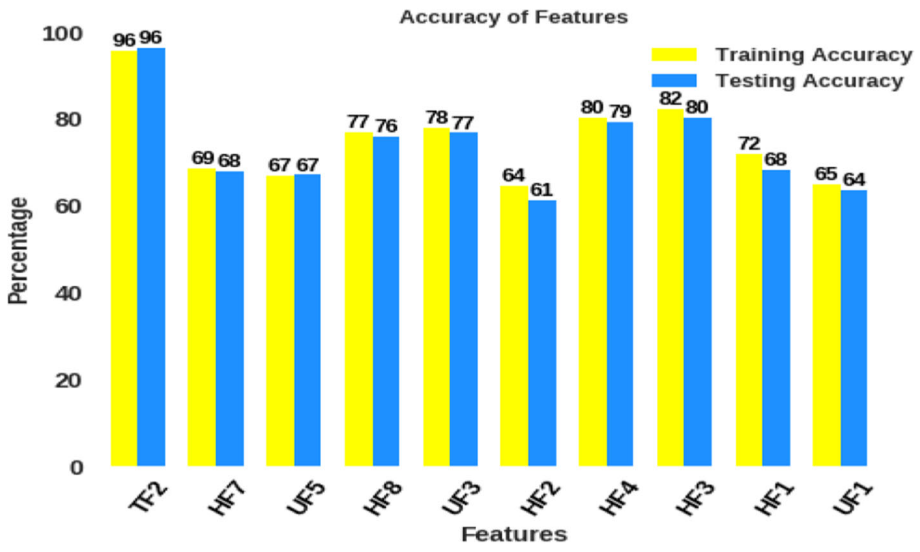

Figure 7. DNN individual feature accuracy.

Table 6. DNN experimental results.

Experiment	α value	No. of features	Epochs	Training accuracy (%)	Testing accuracy (%)
1	0.001	18	5000	98.71	97.95
2	0.001	14	5000	99.96	99.20
3	0.001	10	5000	99.69	98.97
4	0.0001	10	5000	99.51	99.20
	0.0001	10	6000	99.55	99.52

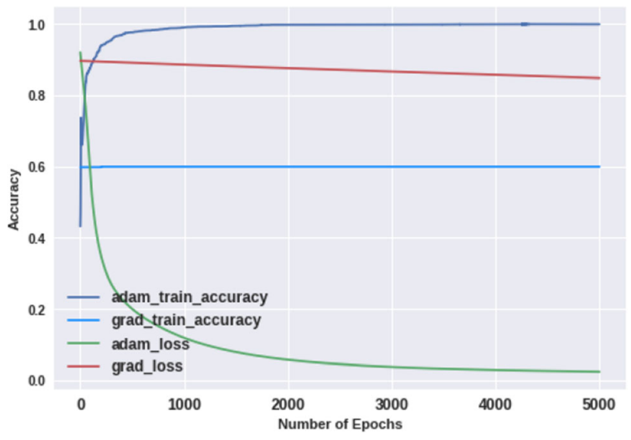

Figure 8. Comparison between optimizers.

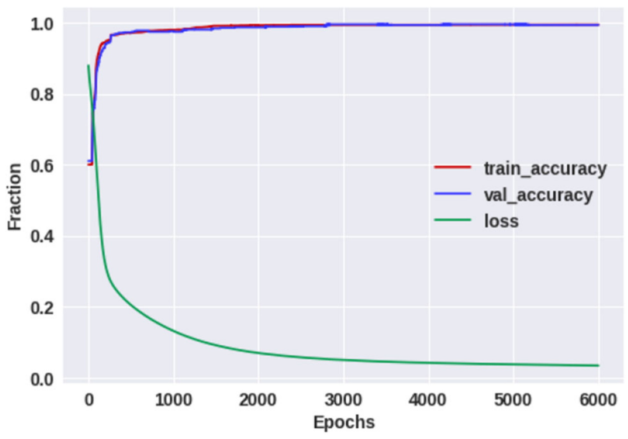

Figure 9. DNN accuracy with ten features.

see in figure 7 at about 800 epochs with the lowest loss (error) of about 0.012%. We achieved a test accuracy of 99.20% and training accuracy of 99.51% with 5000 epochs. We further increased the number of epochs to 6000 by keeping α to 0.0001 and achieved consistent training accuracy of 99.55% and testing accuracy of 99.52%, as shown in figure 9, and all experimental results are tabulated in table 6. We also increased the α to 0.00001, and the loss function converged at much higher epochs with the same accuracy. We, therefore, concluded that further decreasing of alpha would take

Table 7. Parameters for LSTM.

Number of LSTM units	Learning rate	Optimizer	Epochs
4	0.001	Adam Optimizer	700

more processing time without increasing model accuracy.

- *Optimizer*: We used the Adam Optimizer and obtained training accuracy of around 99.55% and testing accuracy of 99.52%. We also tested our model with a gradient descent optimizer, which makes our model very slow and less accurate. This can be clearly seen with the graph shown in figure 8.
- *Number of epochs*: We have used the iterative process to determine the total number of epochs to be used for the best performance of our model. We started with 500 epochs and increased by 500 until we got the minimum loss. The minimum loss means that if we keep iterating the model, the loss will continue to decline to some minimum value and then start fluctuating, so we have to stop at that minimum point. It varies with different learning rates and different optimizers.
- *Number of hidden layers*: Increasing number of hidden layers will result in increased network complexities because we fit our data with the number of hidden layers and the number of hidden units. We have initialized with one hidden layer and moved progressively to identify the optimal hidden layers. Based on the empirical analysis, we achieved the optimal model results with 4 hidden layers. On further increasing the layers, the model showed non-promising results with additional processing time.

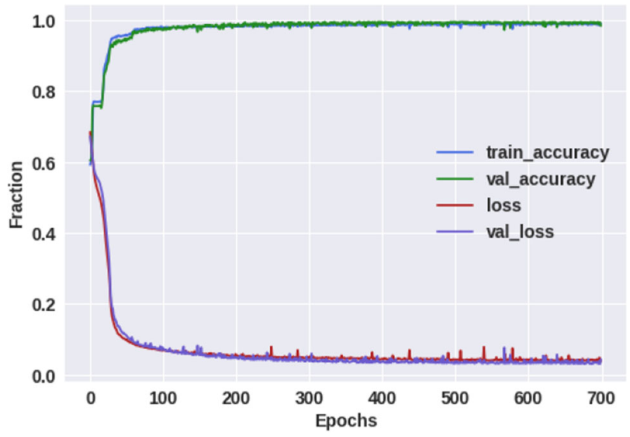

Figure 11. Accuracy graph of LSTM.

- *Number of units in hidden layers*: There is no way to determine the number of hidden units in each layer. Hence we began with a smaller number of hidden units in hidden layers as they were faster, but could not learn properly, and it resulted in less accuracy. Hence, we increased and tested the accuracy of each of these configurations. It is observed that having more units slows down and leads to data over-fitting and therefore lower accuracy is obtained.

Therefore, after all these experiments, we found that DNN is consistent by achieving 99.55% training accuracy and 99.52% testing accuracy with finalized 10 features. The individual features training and testing accuracies of DNN are shown in figure 6. The accuracy graph of selected features using DNN is shown in figure 9. Experimental

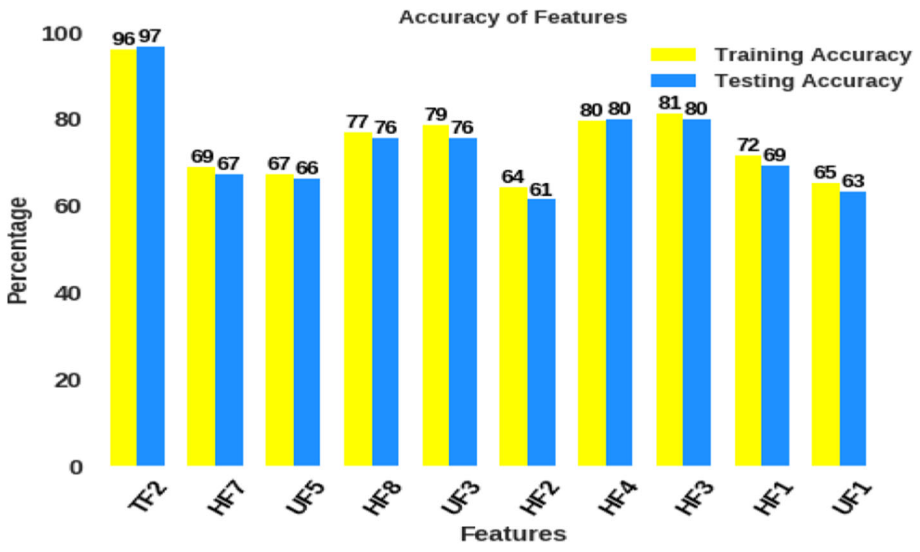

Figure 10. LSTM individual feature accuracy.

Table 8. Parameters for CNN

Layers	Number of filters in layers	Learning rate	Optimizer	Epochs	Window size	Activation function	Stride
7	32, 64, 64, 128, 128, 264, 512	0.001	Adam Optimizer	200	2	tanh	1

results are tabulated in table 6. Due to the close difference between training and test accuracy, over-fitting is reduced.

5.3 *Results with LSTM*

To check the effectiveness of our 10 features, we experimented using LSTM. The parameters used for the experiment are given in table 7. In LSTM, hyperparameters are tuned to achieve better accuracy. These hyperparameters are used in experiment number 5 to achieve promising accuracy.

Experiment 5 – evaluation of LSTM model by tuning parameters:

- Number of LSTM units: We have started with 32 LSTM units. However, more LSTM units slow the processing of training. The network testing accuracy was much lower than the training accuracy, which implies over-fitting. After dropouts were used to reduce over-fitting there was a trade-off between accuracy and over-fitting, and over-fitting was reduced by reducing the precision of training. Hence we started decreasing LSTM units. This trend was seen until the number of LSTM units was reduced to four.
- Learning rate (α): This is one of the most important parameters to determine our model's convergence. If α is kept large (near to one), the minimum convergence point in the model contour can be skipped. If α is kept small (near 0), a long time is required to reach the minimum convergence point. We began with $\alpha=$ 0.0001. However, due to the low magnitude of α, the network was slowly converging even after 5000 epochs, and there was no improvement in accuracy with an increasing number of epochs, and it was 98%. Hence, we increased it to 0.001. At this α, we got our maximum accuracy (99.57%). However we continued till 0.1 (larger α), and it proved to be pointless as the network started oscillating, and accuracy decreased.
- Optimizer: We have experimented with a simple gradient optimizer and applied an incremental approach by adding epochs. The experiment was conducted with a maximum of 10000 epochs, and the convergence rate was going slow with 10000 epochs. In this process, we observed the best possible accuracy with 700 epochs using the Adam Optimizer.

After tuning the afore-mentioned parameters, we achieved a training accuracy of 98.86% and testing accuracy of 99.57%. The accuracy graph that we obtain is shown in figure 10. The individual features testing and training accuracies are given in figure 11. This figure shows that the training and testing accuracies of individual features are very close and prevent over-fitting.

5.4 *Results with CNN*

To verify the performance of our 10 selected features, we have conducted our next experiment using the CNN model.

The parameters used for this experiment are given in table 8. Further details of our experiments are given here.

Experiment 6 – evaluation of CNN model by tuning parameters:

- Window size: It refers to the size of the one-dimensional window, which has to be convolved sequentially. Our dataset has 10 features. The choice for the size of the window was already limited due to less number of features. Hence we started with larger window size (7), and we found out that it did not learn appropriately due to a lesser number of layers (2) by evaluating accuracy and trends of loss values during training. Hence we started reducing the window size, until we reached a window size of 2.
- Stride value: It refers to steps skipped after each convolution. Higher strides will reduce the size of the output. We started with a stride value of 4, and it reduced the number of convolutional layers in the model. However the lower convolutional layer performed poorly. Hence, we reduced it to 1.
- Number of filters in each layer: The value has been summarized in table 8.

Other hyperparameters have been summarized in table 8. It performs better with 10 features, and achieves training accuracy of 99.29% and testing accuracy of 99.43%. The accuracy graph that we have obtained is shown in figure 12.

5.5 *Comparison study*

In this section, we compare our model to those of existing works that use deep learning for the classification of phishing sites. Like other researchers [32, 46–49], the results of existing works are collected from the respective papers for the comparison analysis. The listed results in table 9 are the results obtained by respective authors with

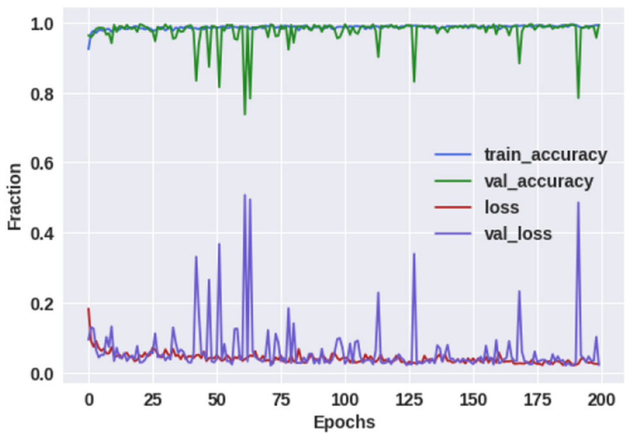

Figure 12. Accuracy graph of CNN.

Table 9. Summary of the results of related existing works.

Techniques	Accuracy (%)
Zhang *et al* [24]	95.83
Mohammad *et al* [35]	92.48
El-Alfy [33]	96.79
Zhao *et al* [34]	98.5
Le *et al* [9]	99.29
Bahnsen *et al* [10]	98.7
Yang *et al* [32]	98.99
Feng *et al* [36]	97.71
Yi *et al* [37]	90

Table 10. Summary of the works implemented on the same dataset.

Techniques	Accuracy (%)
Rao and Pais [1]	99.5
Zhang *et al* [18]	89.18
Xiang *et al* [23]	99.13
Proposed Model-I [DNN]	99.52
Proposed Model-II [LSTM]	99.57
Proposed Model-III [CNN]	99.43

their datasets. These researchers' datasets could not be used for comparison because of the limitation of feature extraction. Our technique requires third-party-based feature (TF2) for the classification of phishing URLs. The use of third-party services for feature extraction requires the datasets with live phishing sites. The existing works (listed in table 9) datasets mostly consist of URLs that have already been taken down from the Internet. Hence, they cannot be used for comparison. The present study has been compared to the previous work [1], CANTINA [18] and CANTINA+ [23] using the common dataset. The results are given in table 10. It is observed that our model with DNN, LSTM and CNN achieved significant accuracy compared with the existing works. It is also demonstrated that the proposed model with LSTM outperforms other proposed models with an accuracy of 99.57%, which is an improvement over the previous work (99.5%) with minimal features. Note that Le *et al* [9], Bahnsen *et al* [10] and Zhao *et al* [34] applied various deep learning algorithms on the URLs rather than content for the classification of phishing URLs. Le *et al* [9] achieved a significant accuracy compared with other existing works that used features extracted from content [24, 33, 35–37]. Despite the use of content-based features in training our model, it is observed that our model outperforms Le *et al* [9] work and other deep-learning-based methods [10, 34], which use URLs for the classification. This shows the richness of our feature set in detecting the phishing sites.

Deployment of model: The model is deployed as a desktop application that takes URL as an input and gives the status of the URL as output. The application makes a connection to REST API, which runs on a remote server where the actual execution of technique takes place. The REST API is hosted on an Intel Xeon 16 core Ubuntu server with 16-GB RAM and a 2.67-GHz processor. The REST API is implemented using the Spring framework, and the GET method is used to transfer the URL from application to the program running on the remote server. On receiving the URL, the running program on the remote server proceeds with the extraction of features. These features are combined to form a feature vector that is further sent to a trained deep learning model for identifying the legitimacy of the given URL. The REST API sends back the status of the URL (legitimate or phishing) to the application to display the message.

6. Limitations

In this section, we discuss the limitations of our proposed work. Since the proposed model is dependent on third-party services, the non-availability of these services will limit the performance of our work.

Also, our proposed model might fail to detect phishing sites that use embedded objects such as flash, java scripts and HTML files to replace textual content. In the future, we intend to include the features for the detection of these embedded objects in the phishing sites.

7. Conclusions

We have proposed a deep learning model to detect the legitimacy of a given website. We used URL heuristic and third-party service-based features for training the deep learning models. Unlike the previous work [1], we minimized the number of features and reduced the dependence on third-party services to achieve a significant accuracy of 99.57%. We also tested our features with various deep-learning-based models such as CNN, DNN and LSTM, and we achieved an accuracy of 99.57% with LSTM, 99.43% with CNN and 99.52% with DNN. The LSTM and DNN outperformed by achieving better results with 10 features than the previous work [1] with machine learning with 18 features.

In the future, we intend to include additional heuristic features that can detect phishing sites hosted on compromised domains, and also phishing sites that include embedded objects such as iframes, flash and HTML.

Acknowledgements

This research was funded by the Ministry of Electronics and Information Technology (MeitY), Government of India. The authors sincerely thank MeitY for financial support. The authors thank the anonymous referees for their comments and criticism, which have helped to improve the quality of the paper.

References

[1] Rao R S and Pais A R 2019 Detection of phishing websites using an efficient feature-based machine learning framework. *Neural Comput. Appl.* 31: 3851–3873
[2] APWG 2018 *Phishing attack trends reports, first quarter 2018*. https://docs.apwg.org//reports/apwg_trends_report_q1_2018.pdf, published July 31, 2018
[3] Fu A Y, Wenyin L and Deng X 2006 Detecting phishing web pages with visual similarity assessment based on earth mover's distance (emd). *IEEE Trans. Dependable Secure Comput.* 3: 301–311
[4] Wenyin L, Huang G, Xiaoyue L, Min Z and Deng X 2005 Detection of phishing webpages based on visual similarity. In: *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, ACM, pp. 1060–1061
[5] Hara M, Yamada A and Miyake Y 2009 Visual similarity-based phishing detection without victim site information. In: *Proceedings of the IEEE Symposium on Computational Intelligence in Cyber Security, CICS'09*, IEEE, pp. 30–36
[6] Rao R S and Ali S T 2015 A computer vision technique to detect phishing attacks. In: *Proceedings of the Fifth International Conference on Communication Systems and network technologies (CSNT)*, IEEE, pp. 596–601
[7] Khonji M, Iraqi Y and Jones A 2013 Phishing detection: a literature survey. *IEEE Commun. Surv. Tutor.* 15: 2091–2121
[8] Zhang N and Yuan Y 2012 *Phishing detection using neural network*. Technical Report, Department of Computer Science, Department of Statistics, Stanford University (CS229 Lecture Notes)
[9] Le H, Pham Q, Sahoo D and Hoi S C 2018 *URLNet: learning a URL representation with deep learning for malicious URL detection.* arXiv preprint: arXiv:180203162
[10] Bahnsen A C, Bohorquez E C, Villegas S, Vargas J and González F A 2017 Classifying phishing URLs using recurrent neural networks. In: *Proceedings of the APWG Symposium on Electronic Crime Research (eCrime)*, IEEE, pp. 1–8
[11] Whittaker C, Ryner B and Nazif M 2010 Large-scale automatic classification of phishing pages. In: *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, vol. 10
[12] Huh J H and Kim H 2011 Phishing detection with popular search engines: simple and effective. In: *Proceedings of the International Symposium on Foundations and Practice of Security*. Springer, pp. 194–207
[13] Jain A K and Gupta B B 2018 Two-level authentication approach to protect from phishing attacks in real time. *J. Ambient Intell. Humaniz. Comput.* 9: 1783–1796
[14] APWG 2014 *Global phishing reports first half 2014*. https://docs.apwg.org//reports/APWG_Global_Phishing_Report_1H_2014.pdf, published 25 September 2014
[15] Cao Y, Han W and Le Y 2008 Anti-phishing based on automated individual white-list. In: *Proceedings of the 4th ACM Workshop on Digital Identity Management*, ACM, pp. 51–60
[16] Zhang J, Porras P A and Ullrich J 2008 Highly predictive blacklisting. In: *Proceedings of the USENIX Security Symposium*, pp. 107–122
[17] Rao R S and Pais A R 2017 An enhanced blacklist method to detect phishing websites. In: *Proceedings of the International Conference on Information Systems Security*. Springer, pp. 323–333
[18] Zhang Y, Hong J I and Cranor L F 2007 Cantina: a content-based approach to detecting phishing web sites. In: *Proceedings of the 16th International Conference on World Wide Web*, ACM, pp. 639–648
[19] Pan Y and Ding X 2006 December Anomaly based web phishing page detection. In: *Proceedings of the 2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*, IEEE, pp. 381–392
[20] Horng M H S, Fan P, Khan M, Run R and Chen J L R 2011 An efficient phishing webpage detector. *Expert Syst. Appl. Int. J.* 38: 12018–12027
[21] Gowtham R and Krishnamurthi I 2014 A comprehensive and efficacious architecture for detecting phishing webpages. *Comput. Secur.* 40: 23–37
[22] Srinivasa Rao R and Pais A R 2017 Detecting phishing websites using automation of human behavior. In: *Proceedings of the 3rd ACM Workshop on Cyber-Physical System Security*, ACM, pp. 33–42
[23] Xiang G, Hong J, Rose C P and Cranor L 2011 Cantina+: a feature-rich machine learning framework for detecting phishing web sites. *ACM Trans. Inf. Syst. Secur.* 14(2): 1–28
[24] Zhang D, Yan Z, Jiang H and Kim T 2014 A domain-feature enhanced classification model for the detection of Chinese phishing e-business websites. *Inf. Manag.* 51: 845–853
[25] Chiew K L, Chang E H and Tiong W K 2015 Utilisation of website logo for phishing detection. *Comput. Secur.* 54: 16–26
[26] Moghimi M and Varjani A Y 2016 New rule-based phishing detection method. *Expert Syst. Appl.* 53: 231–242
[27] Aggarwal A, Rajadesingan A and Kumaraguru P 2012 Phishari: automatic realtime phishing detection on twitter.

In: *Proceedings of the eCrime Researchers Summit (eCrime)*, IEEE, pp. 1–12

[28] Marchal S, Armano G, Gröndahl T, Saari K, Singh N and Asokan N 2017 Off-the-hook: an efficient and usable client-side phishing prevention application. *IEEE Trans. Comput.* 66: 1717–1733

[29] Sahingoz OK, Buber E, Demir O and Diri B 2019 Machine learning based phishing detection from URLs. *Expert Syst. Appl.* 117: 345–357

[30] Li Y, Yang Z, Chen X, Yuan H and Liu W 2019 A stacking model using URL and HTML features for phishing webpage detection. *Future Gener. Comput. Syst.* 94: 27–39

[31] Jain A K and Gupta B B 2018 Towards detection of phishing websites on client-side using machine learning based approach. *Telecommun. Syst.* 68: 687–700

[32] Yang P, Zhao G and Zeng P 2019 Phishing website detection based on multidimensional features driven by deep learning. *IEEE Access* 7: 15196–15209

[33] El-Alfy ESM 2017 Detection of phishing websites based on probabilistic neural networks and *K*-medoids clustering. *Comput. J.* 60: 1745–1759

[34] Zhao J, Wang N, Ma Q and Cheng Z 2018 Classifying malicious URLs using gated recurrent neural networks. In: *Proceedings of the International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. Springer, pp. 385–394

[35] Mohammad R M, Thabtah F and McCluskey L 2014 Predicting phishing websites based on self-structuring neural network. *Neural Comput. Appl.* 25: 443–458

[36] Feng F, Zhou Q, Shen Z, Yang X, Han L and Wang J 2018 The application of a novel neural network in the detection of phishing websites. *J. Ambient Intelli. Humaniz. Comput.* 1–15

[37] Yi P, Guan Y, Zou F, Yao Y, Wang W and Zhu T 2018 Web phishing detection using a deep learning framework. *Wirel. Commun. Mobile Comput.* 2018: Article ID 4678746

[38] Zhou Q, Chen H, Zhao H, Zhang G, Yong J and Shen J 2016 A local field correlated and Monte Carlo based shallow neural network model for non-linear time series prediction. *EAI Endorsed Trans. Scalable Inf. Syst.* 3: e5-1–e5-7

[39] Quinlan J R 1986 Induction of decision trees. *Mach. Learn.* 1:81–106

[40] Smith C and Jin Y 2014 Evolutionary multi-objective generation of recurrent neural network ensembles for time series prediction. *Neurocomputing* 143: 302–311

[41] Mikolov T, Joulin A, Chopra S, Mathieu M and Ranzato M A 2014 *Learning longer memory in recurrent neural networks*. arXiv preprint: arXiv:1412.7753

[42] Jozefowicz R, Zaremba W and Sutskever I 2015 An empirical exploration of recurrent network architectures. In: *Proceedings of the International Conference on Machine Learning*, pp. 2342–2350

[43] Hochreiter S, Schmidhuber J 1997 Long short-term memory. *Neural Comput.* 9: 1735–1780

[44] Krizhevsky A, Sutskever I and Hinton G E 2012 Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems*, pp. 1097–1105

[45] Pham N Q, Kruszewski G and Boleda G 2016 Convolutional neural network language models. In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1153–1162

[46] Ramesh G, Krishnamurthi I and Kumar K S S 2014 An efficacious method for detecting phishing webpages through target domain identification. *Decis. Support Syst.* 61: 12–22

[47] He M, Horng S J, Fan P, Khan M K, Run R S, Lai J L, Chen R J and Sutanto A 2011 An efficient phishing webpage detector. *Expert Syst. Appl.* 38: 12,018–12,027

[48] Marchal S, Armano G, Gröndahl T, Saari K, Singh N and Asokan N 2017 Off-the-hook: an efficient and usable client-side phishing prevention application. *IEEE Trans. Comput.* 66: 1717–1733

[49] Gowtham R and Krishnamurthi I 2014 A comprehensive and efficacious architecture for detecting phishing webpages. *Comput Secur* 40: 23–37