electronics

MDPI

Article

An Effective Phishing Detection Model Based on Character Level Convolutional Neural Network from URL

Ali Aljofey [1,2], Qingshan Jiang [1,*], Qiang Qu [1], Mingqing Huang [1] and Jean-Pierre Niyigena [1]

[1] Shenzhen Key Laboratory for High Performance Data Mining, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China; aljofey@siat.ac.cn (A.A.); qiang@siat.ac.cn (Q.Q.); mq.huang@siat.ac.cn (M.H.); jeanpierre@siat.ac.cn (J.-P.N.)

[2] Shenzhen College of Advanced Technology, University of Chinese Academy of Sciences, Beijing 100049, China

* Correspondence: qs.jiang@siat.ac.cn; Tel.: +86-186-6532-6469

Received: 11 August 2020; Accepted: 4 September 2020; Published: 15 September 2020

Abstract: Phishing is the easiest way to use cybercrime with the aim of enticing people to give accurate information such as account IDs, bank details, and passwords. This type of cyberattack is usually triggered by emails, instant messages, or phone calls. The existing anti-phishing techniques are mainly based on source code features, which require to scrape the content of web pages, and on third-party services which retard the classification process of phishing URLs. Although the machine learning techniques have lately been used to detect phishing, they require essential manual feature engineering and are not an expert at detecting emerging phishing offenses. Due to the recent rapid development of deep learning techniques, many deep learning-based methods have also been introduced to enhance the classification performance. In this paper, a fast deep learning-based solution model, which uses character-level convolutional neural network (CNN) for phishing detection based on the URL of the website, is proposed. The proposed model does not require the retrieval of target website content or the use of any third-party services. It captures information and sequential patterns of URL strings without requiring a prior knowledge about phishing, and then uses the sequential pattern features for fast classification of the actual URL. For evaluations, comparisons are provided between different traditional machine learning models and deep learning models using various feature sets such as hand-crafted, character embedding, character level TF-IDF, and character level count vectors features. According to the experiments, the proposed model achieved an accuracy of 95.02% on our dataset and an accuracy of 98.58%, 95.46%, and 95.22% on benchmark datasets which outperform the existing phishing URL models.

Keywords: phishing detection; URL features engineering; character embedding; deep learning

1. Introduction

Phishing is an offense in which the phisher seeks to trick users into disclosing critical and personal information such as credit card details and passwords. The intent of phishers to carry out a phishing attack is to sell the personality of the victims, to get ransom, to exploit the system's weaknesses, or to receive financial profits [1]. One of these common offenses is to design deceptive sites which are imitations of benign websites (e.g., PayPal, eBay, etc.) and host them in a hacked domain.

It is hard for human eyes to distinguish between benign and deceptive web pages because they look identical. Once the user accesses the imitated site, critical information will be robbed using scripts. Phishing offenses increase every year due to the rapid growth of the e-commerce users. Phishers can use malware (i.e., malicious software), web pages, and emails to carry out phishing offenses.

According to the Anti-Phishing Working Group (APWG), the total number of phishing sites detected by the APWG in the 1st quarter (Q) of 2019 was 180,768, that was up notably from the 138,328 seen in 4Q of 2018, and from the 151,014 seen in 3Q of 2018 [2]. A non-profit industry association has focused on rubbing identity theft and fraud resulting from the phishing raising problem, spyware, and e-mail fraud. Phishing has become a serious problem due to the extensive damage to its target industry e.g., payment, financial institutions, email, etc. The rating of annual direct economic losses to the US economy as result of phishing offenses range from 61 million to 3 billion [3].

As reported by Internet Security Threat Report (ISTR) submitted by Symantec [4], the number of new mobile malware in 2017 grew 54% compared to 2016. The new malware programs are multitasking, they perform many tasks, i.e., downloading and installing other malicious programs without user's permission, theft of critical information, etc. Kaspersky Lab reported that the anti-phishing technique was operated 246,231,645 times in Kaspersky Lab [5].

Some existing techniques use a blacklist to reveal phishing sites. This policy fails to reveal non-blacklisted phishing sites (i.e., zero day attacks). Many heuristic-based detection techniques extract features based on webpage content features and third-party service features to detect phishing sites. However, the use of third-party services such as page rank, network traffic measures, search engine indexing, and WHOIS (domain age) may be restricted to reveal phishing sites hosted on compromised servers and these sites are incorrectly labeled as benign because they are included in the search results. Furthermore, webpage content features and third-party features are time consuming.

Machine learning techniques have also been used to investigate the URL of the web page with different hand-crafted feature sets in order to refine detection efficiency [6,7]. The URLs are first analyzed to perform the feature adaptation from phishing websites. Then, a training set is constructed by machine learning experts using the extracted features along with their labels. Finally, the advantage of classification supervised machine learning algorithms is used to develop a model for phishing detection. However, there are still drawbacks due to the fact that the human effort needs time and additional maintenance labor costs.

Deep learning has also been integrated into phishing detection, driven by recent rapid development and many successful applications [8–10]. Unlike conventional machine learning methods, deep learning techniques implicitly extract hand-crafted features as machine learning specialists can use data directly without the knowledge of cybersecurity experts.

In this paper, we propose a deep learning-based solution for phishing URL detection. Deep learning [11,12] uses layers of stacked nonlinear projections to learn representations of multiple levels of abstraction. It has shown advanced performance in many applications, e.g., natural language processing, computer vision, speech recognition, etc. Specifically, convolutional neural networks (CNNs) have shown auspicious text classification achievement recently [13,14]. After their success, we propose using character level CNN [14] to learn a URL embedding to detect phishing URLs.

The proposed model receives a URL string as an input and applies CNN at the character level in the URL. The model first identifies the individual characters in the training set based on prescribing character vocabulary, and then represents each character as a fixed-length vector using one-hot encoding. Using this, the URL sequence characters are converted into a matrix representation, where the convolution and pooling can be applied. In order to generate the final output, fully connected layers are used to generate an output that depends on the number of classes.

Character level CNN determines important information from specific combinations of characters that appear together which could be symptomatic of wickedness. Basically, a URL is a sequence of characters or words where some words have few semantic meanings. As a result that some URLs contain separable combined words without separators and these combined words may not contain any significance, it is difficult to extract semantic meanings from words in a URL. Furthermore, the phishers may change unnoticeable characters in official website URLs, e.g., "www.icbc.com" as "www.1cbc.com" to make people unable to differentiate the similar view of phishing URLs from benign URLs.

The advantages of the proposed method are listed below:

- Independence of third-party services: Use of third-party services such as web-based blacklist/whitelist, page rank, search engine indexing, network traffic measures, the domain age, etc. raise the efficiency of the detection system. However, these services also raise the detection time, so they cannot be helpful. The proposed method does not rely on any third-party features. It relies only on the URL of the website and the detection time for the URL classification is only 0.47 ms per URL.
- Language independence: The proposed method works effectively for websites with content in all languages because the features are extracted from the URL string and embedded depending on predefined character vocabulary of size M for the input language.
- New websites detection: Due to character level embedding features, character-level embedding for new URLs can be easily generalized. The proposed model can detect new phishing sites that have not been classified as fraudulent phishing before. Using this feature, the proposed method is sturdy in the zero-day attack, which is one of the most serious types of phishing offenses.
- Independence of cybersecurity experts: The required expert features engineering is reduced, as CNN automatically recognizes features to represent the URL not relying on any other complex or expert features during the learning task.

The main contributions of this paper are summarized as follows:

- This paper proposes a phishing detection model with a deep learning-based solution, which can speedily and precisely detect phishing sites using only URL features of the webpage.
- We define four different types of feature sets as hand-crafted, character embedding, character level TF-IDF, and character level count vectors features, and then compare different machine and deep learning algorithms using these various feature sets in order to measure the efficiency.
- We use four datasets D1, D2, D3, and D4 to evaluate the performance of the proposed model, where D1 is our dataset consisting of 318,642 empirical URLs and D2, D3, D4 are used in existing baseline works. The accuracy of the experimental results was 95.02% on our dataset, and 98.58%, 95.46%, 95.22% on benchmark datasets, which is much better than the existing phishing URL models.

The remainder of the paper is organized as follows: Section 2 reviews related works about existing phishing detection techniques. In Section 3, the proposed work is introduced in detail. Section 4 conducts different comparative experiments and discusses the results. Finally, the summary and future works of the paper are presented in Section 5.

2. Related Work

Generally, phishing can be detected through list-based detection, machine learning-based detection, heuristic-based detection, or deep learning-based detection methods. However, the problem of phishing is so complicated that there is no decisive solution to effectively override all threats; therefore, multiple techniques are often involved to prevent specific attacks. The protection methods are categorized into two main groups: expanding the user knowledge and using some additional software as described in Figure 1.

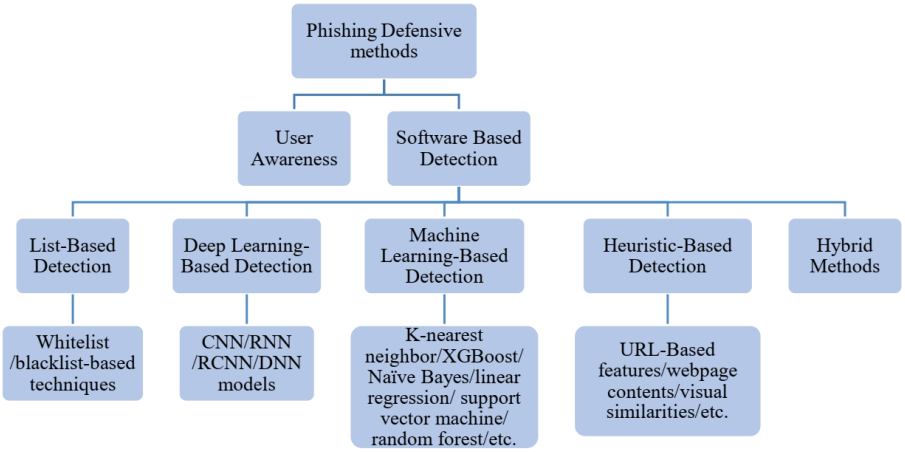

Figure 1. Overview of phishing detection techniques.

2.1. List-Based Detection

List-based phishing detection techniques can be classified into whitelist and blacklist-based techniques. The whitelist is a list of benign URLs and IP addresses used to validate a suspicious URL. Wang et al. [15], Chou et al. [16], and Han et al. [17] use a whitelist-based technique to detect phishing URLs.

Blacklist-based techniques are vastly used in overtly available anti-phishing toolbars e.g., Google safe browsing, which checks URLs versus Google's periodically updated blacklist of browser phishing sites and gives warnings to users once a URL is counted as phishing. Although these list-based methods can be comparatively high precision, it is hard to keep a comprehensive list of phishing URLs since new URLs are created on a daily basis.

In Felegyhazi et al. [18], domain name and server information of blacklisted URLs are used to detect new phishing URLs where registration information and Domain Name System (DNS) zone of new phishing URLs are compared to the information of blacklisted URLs. Rao and Pais [19] developed an improved blacklisted method for detecting phishing URLs. Instead of using blacklist of URLs, a blacklist of signatures is generated using content-based features. Prakash et al. [20] developed an enhanced two-component method to block hackers from avoiding the blacklist detection. One of the components is used to extend the blacklist through simple sets of blacklist phishing sites using five heuristics elements (i.e., the top-level range, the IP address, the directory structure, the query string, and the brand name), whereas the other component is used to make an approximate match to a particular URL to determine whether the phishing URL is a phishing one. However, list-based techniques cannot catch zero-day attacks, which means that they can only react to the newly unexpected phishing URLs when the developers update the blacklist.

2.2. Heuristic-Based Detection

The essence of the heuristic-based detection techniques, which have been developed from list-based detection techniques, is the creation of phishing site characteristics based on many hand-crafted features, for example, URL-based features, webpage contents, website visual similarities, etc.

Zhang et al. [21] proposed a method of detecting phishing called Cantina based on Google search engine to retrieve keywords and domain names in a webpage. It uses the results returned by research and other heuristic rules to determine whether the webpage is legitimate or phishing. However, the method is responsive only to the English language. Another version of previous work called Cantina+ was proposed by Xiang et al. [22]. It takes 15 heuristic features as the input to train a classifier for detection of phishing sites. Ramesh et al. [23] proposed a method to detect phishing sites by checking web pages and identifying all direct or indirect links to the web pages. Although the method provides high detection accuracy, it is time consuming because it relies on search engines and

third-party services such as DNS query. A phishing detection algorithm (PDA) is proposed by Jain and Gupta [24] to decide whether a suspicious URL is a phishing website by calculating the number of hyperlinks in the suspicious webpage. The study gives a true positive test result of 86.07% and a false negative of 1.48% on 405 benign pages and 1120 phishing pages, which indicates that the method has failed to attain the expected level of efficiency.

2.3. Machine Learning-Based Detection

While more information can improve the accuracy of phishing detection, it is often not feasible to obtain a lot of features because of the limited time and computation resources available. To deal with these limitations, in recent years phishing detection has received much research using machine learning techniques (e.g., K-nearest neighbor, XGBoost, Naïve Bayes, linear regression, support vector machine, and random forest).

In Zhang et al. [25], the features are extracted from URLs based on bag-of-words and then trained by a classifier through online learning. Similarly, in Sahingoz et al. [7], distributed representations of words are adopted in a particular URL, and then seven different machine learning algorithms are used to predict whether the URL is a phishing URL. Although these methods have shown satisfying performances, they cannot deal with unseen information that does not exist in the training set.

Mogimi et al. [26] proposed a phish detector, in which support vector machine (SVM) algorithm is used first to train a phishing detection model, and then the decision tree (DT) algorithm is used to extract the hidden phishing. The true positive and false negative of the proposed method are 0.99 and 0.001, respectively in a large dataset. However, this method supposes that phishing web pages only use benign page content, which does not apply in practice. Recently, Rao et al. [6] proposed a light-weight application, CatchPhish which predicts the URL legitimacy without visiting the content of the website. The proposed model extracts hand-crafted and Term Frequency-Inverse Document Frequency (TF-IDF) features from the suspicious URL for classification using the random forest classifier.

2.4. Deep Learning-Based Detection

Due to the success of the Natural Language Processing (NLP) achieved by deep learning techniques, some of them have recently been employed for phishing detection e.g., CNN, recurrent neural network (RNN), recurrent convolutional neural networks (RCNN), and deep neural network (DNN). Although deep learning techniques are not exploited much in phishing detection due to the extensive training time, they often provide more accuracy and automatically extract the features from raw data without any prior knowledge.

Wang et al. [8] proposed a fast phishing website detection method called precise phishing detection with recurrent convolutional neural networks (PDRCNN) that depends only on the URL of the website. It encodes the information of a URL into a two-dimensional tensor and feeds the tensor into a deep learning neural network to classify the original URL. They use first a bidirectional long short-term memory (LSTM) network, and then a convolutional neural network (CNN) to extract global and local features of the URL. YANG et al. [9] developed a multidimensional feature phishing detection method based on two steps. In the first step, character sequence features of the given URL are extracted and used for classification by LSTM-CNN deep learning networks. In the second step, URL statistical features, webpage content features, and the classification result of deep learning are combined into multidimensional features. Wei et al. [27] present a light-weight deep learning algorithm. They use a novel character-level multi-spatial deep learning model to detect phishing URLs.

Yuan et al. [10] developed a method that combines character embedding (word2vec) with the structures of URLs to obtain the URLs vector representations. They partition the URL into five parts: URL protocol, sub-domain name, domain name, domain suffix, and URL path. The vector representations of URLs are trained by existing classification algorithms to identify the phishing URLs. Le et al. [28] developed a URLNet method for malicious website URL detection. In this method, character-level and word level features are extracted based on URL strings and CNN network is used

for training and testing. Huang et al. [29] proposed the PhishingNet deep learning-based method for detection of phishing URLs. They use a CNN network to extract character-level features of URLs; meanwhile, they employ an attention-based hierarchical recurrent neural network (RNN) to extract word-level features of URLs. After that, they fuse and train these features through three convolutional layers and two fully connected layers. Bahnsen et al. [30] compared a random forest classifier against recurrent neural networks based on a hand-crafted features method. The recurrent neural network model achieved an accuracy of 98.7%, which is 5% higher than the random forest classifier without the need of manual feature creation.

2.5. Hybrid Method-Based Detection

Hybrid detection techniques rely on combining more than one of the previous techniques in order to achieve good performance in the detection of phishing sites. Yang et al. [9] propose a multidimensional feature phishing detection approach which consists of two steps. In the first step deep learning algorithms (CNN-LSTM) are used to extract URL features. In the second step, they combine URL statistical features, webpage code features, webpage text features, and the classification result of deep learning into multidimensional features, which are then classified by a machine learning algorithm (XGBoost).

In Sahingoz et al. [7] and Rao et al. [6], heuristic-based features are extracted from different parts of the URL and fed to a machine learning algorithm i.e., random forest (RF) to reveal the legitimacy of the URL. Jain et al. [31] present a two-level validation approach of phishing detection using third-party services and webpage contents. In the first level of validation, the search engine-based technique is proposed which uses a simple query to validate the webpage. The second level of validation processes the validity of various hyperlinks within the source code of the web page to detect phishing websites.

In this paper, we propose a convolutional neural network (CNN) to extract features automatically from only the URL without using any manually designated features by humans. Character level CNN is used instead of word level due to the difficulty of extracting semantic meanings from words in a URL where the URL typically contains meaningless combined words.

3. Proposed Methodologies

In general, attackers intend to create phishing URLs so that they appear as legitimate websites to the users. Attackers use different URL jamming techniques to trick users into revealing personal information that can be misused. The main idea of this paper is to quickly detect phishing websites using lightweight features. This is achieved by extracting only features from the URL without visiting the website content. Before going to the proposed model architecture, a brief description of the component of URLs is debated in the section below.

3.1. URL Components

The URL refers to the resource locator. It is used for locating a resource on the web such as hypertext pages, images, files, and audio. Figure 2 illustrates the different components of a URL with an example.

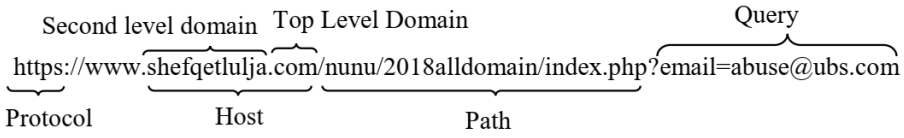

Figure 2. URL components.

The first component of the URL is a protocol (https, http, ftp, etc.) that is an established set of regulations that governs how data will be transmitted from source to destination. The second component refers to the host IP address or resource's location. The host name is divided into primary

domain and Top Level Domain (TLD). The primary domain and TLD together represent the host name of the URL. The host name is followed by a port number that is an optional field. In the third component, the path is used to recognize the particular resource within the domain requested by the user. The path is followed by optional field such as query. The base URL is concatenated by protocol, hostname, and URL path.

Although the second level domain name commonly shows the type of site or company name, the attacker can easily find or purchase it for phishing. The second domain name can be specified only once at the inception. However, unlimited number of URLs can be created by an attacker who has expanded the second level by file and path names, because the internal address design depends directly on the phishers. The unique part of the URL is the composition of the second domain and top level domain names, which are called the host domain. Thus, cybersecurity companies are making a great effort to identify the fraudster domains by name, which are used for phishing offenses. If a hostname is specified as phishing, an IP address can be simply blocked to prevent it from obtaining the web pages existing in it.

3.2. Model Design

In this section, we present the details of the proposed model configuration. The implicit deep neural network for the proposed model is a convolutional neural network (CNN). The CNNs are used to learn sequential information from the URL. Particularly, CNNs are applied at the character-level. A URL u is basically a concatenation of characters. We aim to get a matrix representation $u \rightarrow G \in \mathbb{R}^{L \times k}$, so that the instance G includes a set of adjacent components $g_i, i = 1, \ldots, L$ in a concatenation. Each k-dimensional vector is depicted by an embedding such that $g_i \in \mathbb{R}^k$. Normally, this dimensional depiction of the component is the embedding vector extracted from the embedding matrix that is initialized manually. An instance can be depicted in concatenation of L components as follows [28]:

$$G = G_{1:L} = g_1 \oplus g_2 \oplus ... \oplus g_L \tag{1}$$

where $\oplus$ refers to the concatenation operator. Usually all sequences are filled as 0 or amputated to the same length L. The CNN network will convolve up via this instance $G \in \mathbb{R}^{L \times K}$ using a convolution operator. The h-length convolution consists of convolving a filter $X \in \mathbb{R}^{k \times h}$ followed by a non-linear activation f (i.e., rectified linear units) to generate a new feature:

$$y_i = f\left(X \otimes G_{i:i+h-1} + b_i\right) \tag{2}$$

where b_i is the bias.

The output of this convolution layer applies the X filter with a nonlinear activation for each h-length portion of its inputs separated by a predetermined stride value. These outputs are then concatenated to generate output Y as follows:

$$Y = [y_1, y_2, \ldots, y_{L-h+1}] \tag{3}$$

After convolution, the pooling step (temporal max-pooling or average pooling) is applied to trim the dimension of the feature, to determine the most important features, and to train deeper models. Using a filter X to convolve each portion of length h, the CNN is able to take advantage of the temporal relationship of length h in its input. The CNN model usually consists of multiple groups of filters with different lengths (h), and each group consists of several filters.

The convolution that the pooling layer follows is made up of a block in this deep neural network. There can be many of these blocks that can be stacked on top of each other. The pooled features of the final block are grouped and passed to fully connected (FC) layers for classification purpose. The algorithm stochastic gradient descent (SGD) can then be used to train the network, where the gradients are obtained by back-propagation [32] to carry out optimization.

More specifically, the URL is taken as a raw input and indexed according to the Table 1. We can see, besides the 94 characters, we also have an unknown token(UNK) to represent the rare characters in vocabulary. After that, the URL is settled into fixed-size sequence by truncating or zero-filling, and one-hot vector is then used to represent these 95 words, which means that each character has 95 dimensions. The URL features are extracted and reduced from the embedding matrix via the convolutional and max pooling layers. However, to generate the final output, the two fully connected layers receive the result of the pooling to generate an output equal to the number of classes. An overview can be seen in Figure 3.

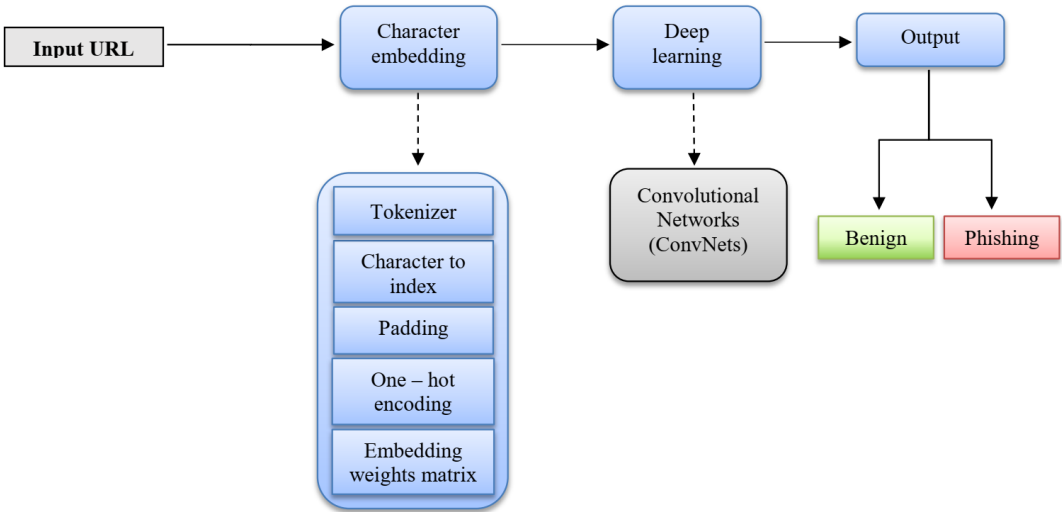

Figure 3. An overview of the proposed model.

3.2.1. Character Level Embedding Features

We employ the method used in [14] to represent and embed the URL with expanding of the character vocabulary used. The processing of URL at the character level is a solution to a problem out of vocabulary. Character-level embedding is used instead of word-level embedding because URLs typically use words without any significance.

More information is included at the character level. Attackers also simulate the URLs of original websites by changing several unnoticeable characters. For example, they might change google.com to g00gle.com, replace "oo" with "00". Character-level embedding helps to find this imitative information, and improve the performance of detecting malicious URLs.

At the embedding stage of the URL, embedding is done by determined an m-sized alphabet for the input language, then embedding each character using one-hot encoding (see Figure 4). After that, the sequence of characters is converted into a sequence of these m-sized vectors with a fixed-length L. Any character overriding length L is truncated, and any character that is not in the alphabet including blank character is embedded as all-zero vectors.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}$$

Figure 4. Example of one hot-encoding.

The main processes of character level embedding are:

- Tokenizer: The tokenizer is used to process URL in char level and add a UNK token to the vocabulary. After fitting the training data, the tokenizer will contain all the necessary information about the data.
- The vocabulary: The alphabet used consists of 95 characters, including 26 lower-case English letters, 26 upper-case English letters, 10 numbers, and 33 other characters (e.g., ,;.!?: ' /_@#$...etc.) as illustrated in Table 1.
- Character to index: After getting the right vocabulary according to Table 1, we can represent all URLs by using character index as shown in Figures 5 and 6.
- Padding: URL has a different length and the neural network can only handle fixed-length vectors, therefore all URLs must be of equal length so the CNN can process the batch data. Here, we define the maximum length of URL as 200. If the length of the URL is smaller than 200, the remaining part will be filled as 0. If the length of the URL is bigger than 200, the part longer will be amputated. Therefore, all URLs will preserve the same length.

Table 1. Character dictionary.

Character	Encode
abcdefghijklmnopqrstuvwxyz	1–26
ABCDEFGHIJKLMNOPQRSTUVWXYZ	27–52
0123456789	53–62
,;.!?:"'/\\\|_@#$%ˆ&*~`+-=<>()[]{}	63–94
UNK	95
Default	0

```
l2delink.lt/2017/05/
[12, 55, 4, 5, 12, 9, 14, 11, 65, 12, 20, 71, 55, 53, 54, 60, 71, 53, 58, 71]
```

Figure 5. URL index representation example.

```
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7, 'h': 8, 'i': 9, 'j': 10, 'k': 11, 'l': 12, 'm': 13, 'n': 14, 'o': 15, 'p': 16, 'q': 17, 'r': 18, 's': 19, 't': 20, 'u': 21, 'v': 22, 'w': 23, 'x': 24, 'y': 25, 'z': 26, 'A': 27, 'B': 28, 'C': 29, 'D': 30, 'E': 31, 'F': 32, 'G': 33, 'H': 34, 'I': 35, 'J': 36, 'K': 37, 'L': 38, 'M': 39, 'N': 40, 'O': 41, 'P': 42, 'Q': 43, 'R': 44, 'S': 45, 'T': 46, 'U': 47, 'V': 48, 'W': 49, 'X': 50, 'Y': 51, 'Z': 52, '0': 53, '1': 54, '2': 55, '3': 56, '4': 57, '5': 58, '6': 59, '7': 60, '8': 61, '9': 62, ',': 63, ';': 64, '.': 65, '!': 66, '?': 67, ':': 68, '"': 69, "'": 70, '/': 71, '\\': 72, '|': 73, '_': 74, '@': 75, '#': 76, '$': 77, '%': 78, '^': 79, '&': 80, '*': 81, '~': 82, '`': 83, '+': 84, '-': 85, '=': 86, '<': 87, '>': 88, '(': 89, ')': 90, '[': 91, ']': 92, '{': 93, '}': 94, 'UNK': 95}
```

Figure 6. Character vocabulary index representation example.

3.2.2. Structure of the Convolutional Neural Networks

As shown in Figure 7, the proposed convolutional neural network (CNN) model has the following layers: (1) embedding layer; (2) convolutional layers; (3) fully connected layers; (4) output layer.

Embedding layer: The embedding layer is generally used in the first layer of the CNN structure for a Natural Language Processing (NLP) problem. In addition to tokenization, the sparse one-hot matrix is reduced and converted into a dense character through the embedding layer. The multiplier of the vector that is returned from the embedding layer can indicate the relations between characters, which can help improve the performance.

Convolutional layers: Following the embedding layer, 7 convolutional layers are used. The convolutional filters (i.e., filter number, filter size or kernel, and pooling size) are applied for each convolutional layer to extract the most valuable features and remove unnecessary learning features. Following each convolutional layer, the rectified linear unit (ReLU) activation function is used. Then, the output of the convolutional layers is flattened.

Fully connected layers: Three fully connected (FC) layers are used to analyze the local deep association features from the convolutional and max pooling layers, each FC layer is followed by one ReLU activation function, and 2 dropout modules in between the 3 fully connected layers are used to prevent overfitting.

Output layer: The number of output units for the last FC layer is set to 2. The softmax function is used in this layer to identify the possibility that the URL belongs to a phishing website, whereas the output range of softmax function is between 0 and 1.

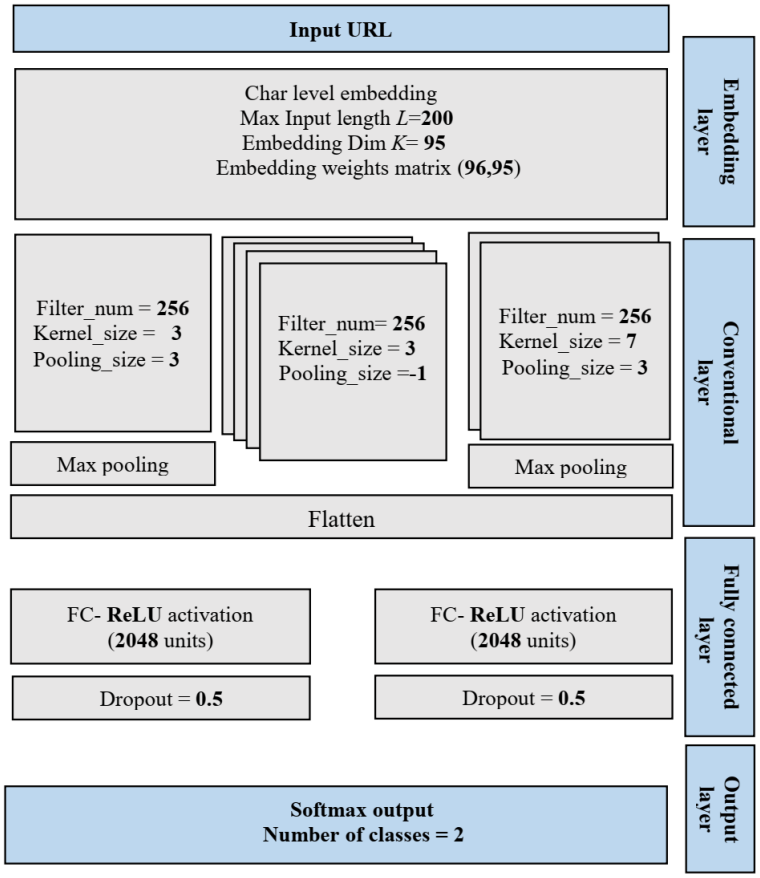

Figure 7. Configuration of the convolutional neural network (CNN).

3.3. URL Features

We used different types of URL features to measure the performance of the proposed model. The URL-based extracted features are categorized into four groups:

- Character embedding level features.
- Character level TF-IDF features.
- Hand-crafted features.
- Character level count vectors features.

We discussed the character embedding level features in Section 3.2.1.

3.3.1. Character Level TF-IDF Features

TF-IDF stands for Term Frequency-Inverse Document Frequency. TF-IDF score represents the relative importance of a term in the document and the entire corpus. TF-IDF score is composed of two terms: the first computes the normalized Term Frequency (TF), the second term is the Inverse Document Frequency (IDF), computed as the logarithm of the number of the documents in the corpus divided by the number of documents where the specific term appears [33].

$$TF\,(t,d) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document\ d}{Total\ number\ of\ terms\ in\ the\ document} \tag{4}$$

$$IDF(t,D) = \log_e\left(\frac{Total\ number\ of\ documents\ D}{Number\ of\ documents\ with\ term\ t\ in\ it}\right) \tag{5}$$

$$TF\text{-}IDF\,(t,d,D) = TF\,(t,d) * IDF\,(t,D) \tag{6}$$

TF-IDF Vectors can be generated at different levels of input tokens (words, characters, n-grams):

- Word level TF-IDF: Matrix representing TF-IDF scores of every term in different documents.
- Character level TF-IDF: Matrix representing TF-IDF scores of character level n-grams in the corpus.
- N-gram level TF-IDF: N-grams are the combination of N terms together. This matrix represents TF-IDF scores of N-grams.

It is noted that TF-IDF has been applied in several works to detect phishing of websites by inspecting URLs [6], to get the indirect associated links [34], target website [22], and legitimacy of suspicious website [21]. Although TF-IDF extracts eminent keywords from the textual content, it has some restrictions. One of the restrictions is that the technique fails when misspelled keywords are extracted. Since URL might contain meaningless words, we applied character level TF-IDF technique with max features as 5000.

3.3.2. Hand-Crafted Features

These are manually crafted features obtained from Python when a URL is provided as input. Depending on the URL parts (host name, path, file, or query) used to extract features, hand-crafted features are categorized into five groups as follows:

- Full URL-based features.
- Domain-based features.
- Path-based features.
- File-based features.
- Query-based features.

Table 2 provides a list of URL hand-crafted features. Most of these features are taken from existing works [6,35–39] and classified into three categories.

Count-based features: These features get the number of specific characters in the URL. The importance of these features is that, if the number or length is higher, it denotes that entered URL is probably a phishing URL. For example, take into consideration the feature that gets the count of dots in the URL. The higher number of dots, the more likely that URL is deceptive as the dots are used to hide the brand name in the URL. U1, U17∼U18, U20, H1, H17∼ H18, P1, P17∼P18, F1, F17∼F18, Q1, and Q17∼Q19 features fall under this category.

Infrequent symbols in benign sites but frequent in phishing sites: There are many special characters (e.g., -, ., " ", /, ?, =, @, &, !, ", ",", +, *, #, $, %) that appear frequently in the URL of phishing sites but not in benign sites. An unusual amount of these characters can indicate the presence of a malicious URL. For example, take into consideration the feature that detects the presence of @ in the URL. If it exists, it could be phishing site otherwise a benign site. Features U10, U2∼U16, H2, H4∼H16, P2∼P16, F2∼F16, and Q2∼Q16 fall under this category.

Other features: Some of the other features do not fall in any of the previous categories. Feature U19 checks if there is an email in URL is set to true; otherwise, is set to false. Features U20 and Q20 check the amount of top-level domains (TLD) present in the URL. Top-level domains are obtained from Internet Assigned Numbers Authority (IANA) (https://www.icann.org/resources/pages/tlds-2012-02-25-en)

and stored in a file. A list of all valid top-level domains is maintained by the IANA and is updated from time to time. Feature H19 checks if the domain has an IP address. Feature H3 checks the total count of vowels in domain name. Based on our dataset, we observed that there are more vowels in phishing URLs than in benign URLs.

Table 2. List of hand-crafted features.

Full URL-Based Features	Domain-Based Features	Path-Based Features	File-Based Features	Query-Based Features
U1: dot_url	H1: dot_host	P1: dot_path	F1: dot_file	Q1: dot_params
U2: hyphe_url	H2: hyphe_host	P2: hyphe_path	F2: hyphe_file	Q2: hyphe_params
U3: underline_url	H3: vowels_host	P3: underline_path	F3: underline_file	Q3: underline_params
U4: bar_url	H4: underline_host	P4: bar_path	F4: bar_file	Q4: bar_params
U5: question_url	H5: bar_host	P5: question_path	F5: question_file	Q5: question_params
U6: equal_url	H6: question_host	P6: equal_path	F6: equal_file	Q6: equal_params
U7: arroba_url	H7: arroba_host	P7: arroba_path	F7: arroba_file	Q7: arroba_params
U8: ampersand_url	H8: ampersand_host	P8: ampersand_path	F8: ampersand_file	Q8: ampersand_params
U9: exclamation_url	H9: exclamation_host	P9: exclamation_path	F9: exclamation_file	Q9: exclamation_params
U10: white_space_url	H10: equal_host	P10: white_space_path	F10: white_space_file	Q10: white_space_params
U11: plus_url	H11: white_space_host	P11: til_path	F11: til_file	Q11: til_params
U12: til_url	H12: til_host	P12: comma_path	F12: comma_file	Q12: comma_params
U13: comma_url	H13: comma_host	P13: plus_path	F13: plus_file	Q13: plus_params
U14: asterisk_url	H14: asterisk_host	P14: asterisk_path	F14: asterisk_file	Q14: asterisk_params
U15: hashtag_url	H15: hashtag_host	P15: hashtag_path	F15: hashtag_file	Q15: hashtag_params
U16: money_sign_url	H16: money_sign_host	P16: money_sign_path	F16: money_sign_file	Q16: money_sign_params
U17: percentage_url	H17: percentage_host	P17: percentage_path	F17: percentage_file	Q17: percentage_params
U18: len_url	H18: len_host	P18: len_path	F18: len_file	Q18: len_params
U19: email_exist	H19: ip_exist			Q19: number_params
U20: count_tld_url				Q20: tld_params

Figures 8 and 9 illustrate a comparison between fishing and benign hand-crafted features of URL based on the average occurrence rate per feature within each URL in our dataset.

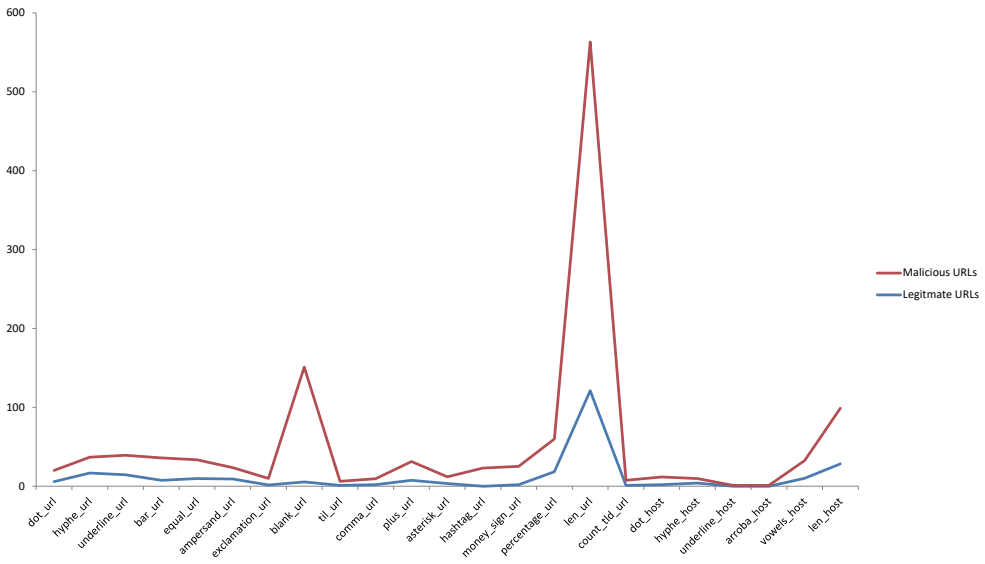

Figure 8. Distribution of URL hand-crafted features in our dataset.

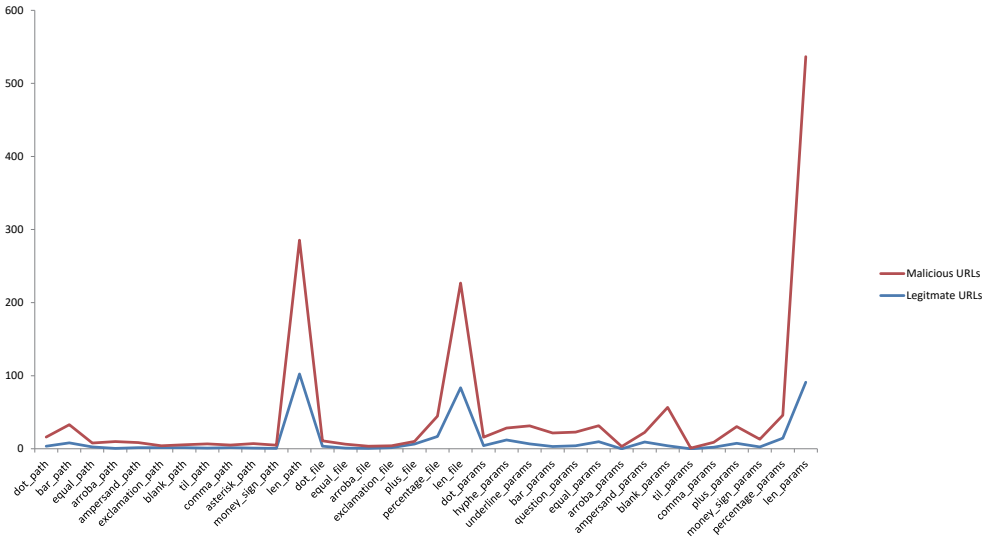

Figure 9. Distribution of URL hand-crafted features in our dataset.

3.3.3. Count Vectors Features

Count vector is a matrix notation of the dataset in which every row represents a document from the corpus, every column represents a term from the corpus, and every cell represents the frequency count of a particular term in a particular document [33]. The basic count vectors (bag-of-words) technique does not consider the meaning of the word in the document. It completely ignores the context in which it is used. The same word can be used in multiple places based on the context or nearby words. For the large document, the vector size can be huge resulting in a lot of computation and time. Here, we applied this technique at the character level instead of word level on URLs corpus.

3.4. Classification Algorithms

To evaluate the performance of URL features, we applied different classification models (Naïve Bayes, Logistic Regression, random forest, XGBoost, and deep neural networks). The main objective of comparing different models is to choose the best model. To implement many machine and deep learning models, the Scikit-learn (http://scikit-learn.org) and Keras (https://keras.io) packages are used.

The Naïve Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naïve Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Naïve Bayes model is easy to build and particularly useful for very large datasets [40]. Equation (7) provides a way of calculating posterior probability $P(c|x)$ from $P(c)$, $p(x)$ and $P(x|c)$.

$$P(c|x) = \frac{P(x|c)\,P(c)}{P(x)} \tag{7}$$

where:

P(c | x) is the posterior probability of class (c, target) given predictor (x, attributes).
P(c) is the prior probability of class.
P(x | c) is the likelihood which is the probability of predictor given class.
P(x) is the prior probability of predictor.

Logistic regression classification is an estimation of logit function that measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic/sigmoid function. Logit function is simply a log of odds in favor of the

event. This function creates a s-shaped curve with the probability estimate [41]. Figure 10 shows the definition of logit function.

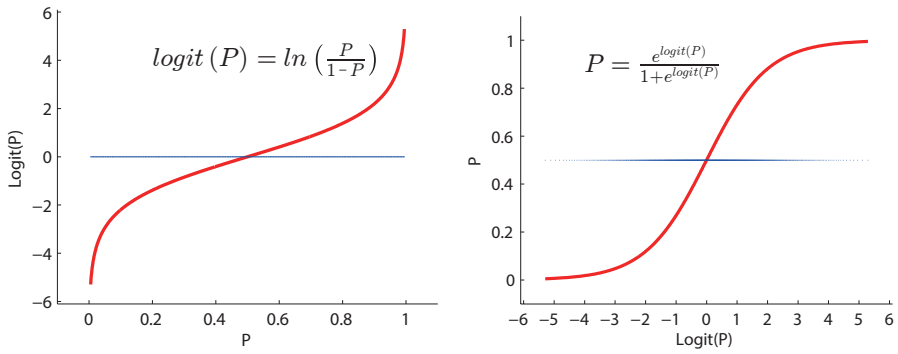

Figure 10. Logit function.

Random forest models are a type of ensemble models, particularly bagging models. They are part of the tree-based model family. It is like the bootstrapping algorithm with decision tree (CART) model. Suppose we have 1000 observation in the complete population with 10 variables. Random forest tries to build multiple CART models with different samples and different initial variables. For instance, it will take a random sample of 100 observation and 5 randomly chosen initial variables to build a CART model. It will repeat the process (suppose) 10 times and then make a final prediction on each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction [42].

Boosting models are another type of ensemble model part of tree-based models. Boosting is a machine learning ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones. A weak learner is defined to be a classifier that is only slightly correlated with the true classification [33]. XGBoost algorithm is a type of boosting models and short form for eXtreme Gradient Boosting. It has both a linear model solver and tree learning algorithms. What makes it fast is its capacity to do parallel computation on a single machine [43].

A neural network is a mathematical model that is designed to behave similar to biological neurons and nervous system. These models are used to recognize complex patterns and relationships that exists within labeled data. Deep neural networks are more complex neural networks in which the hidden layers perform much more complex operations than simple sigmoid or ReLU activations [33]. Figure 11 shows the structure of deep neural networks.

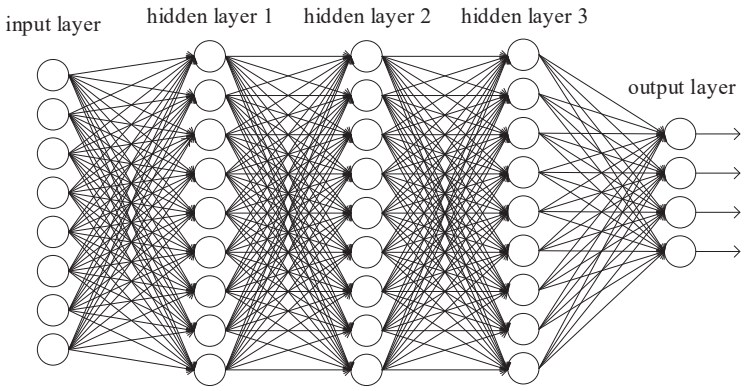

Figure 11. Deep neural networks structure.

4. Experimentation and Result Analysis

The proposed model receives suspicious URLs as an input and generates the URL status as benign or phishing. This section provides empirical details of the proposed model's classification algorithms and detailed feature extraction types are used. Then, comparative test results between these algorithms are portrayed with related features.

The extracted features are divided into four different feature groups as described below:

- FG1: Character level TF-IDF features.
- FG2: Character embedding level features.
- FG3: Character level count vectors features.
- FG4: Hand-crafted features.

4.1. Dataset Description

We collected URLs from different sources (Alexa [44], openphish [45], spamhaus.org [46], techhelplist.com [47], isc.sans.edu [48], and phishtank [49]). Before using them, URLs that are repeated or not surviving were removed to construct a dataset. Distribution of the legitimate and phishing URLs are shown in Table 3. The common URL parts, such as "http://", "https://", and "www." are removed. Without trimming the prefixes, inconsistent URL formats can easily affect the quality of the model during training. The dataset was randomly split into a training set and a test set. The database management system (i.e., pgAdmin) has been used with python to import and pre-process the data.

We divide the dataset into four groups where D1 is our dataset and D2, D3, D4 are datasets used in situated literature as shown below:

- D2: Legitimate sites from Yandex and phishing sites from PhishTank [7].
- D3: Legitimate sites from common-crawl (http://index.commoncrawl.org/) and phishing sites from PhishTank [6].
- D4: Legitimate sites from both common-crawl and Alexa database and phishing sites from PhishTank [6].

Table 3. Dataset distributions.

Dataset	Benign URLs	PhishingURLs	Total
Data 1	157,626	161,016	318,642
Data 2	36,400	37,175	73,575
Data 3	43,189	40,668	83,857
Data 4	42,220	40,668	82,888

The distributions of URL length in dataset D1 for both benign and phishing URLs are shown in Figures 12 and 13, respectively. As can be seen from the figures, the length of URL ranges from 10 to 80 for most URLs, the maximum length for benign URLs is 400, and the maximum length for phishing URLs is 6000.

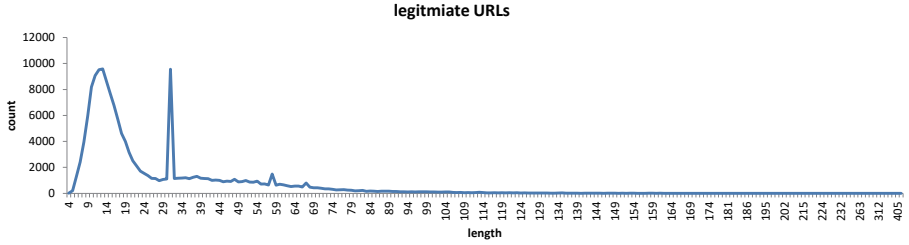

Figure 12. Distribution of URL length in D1.

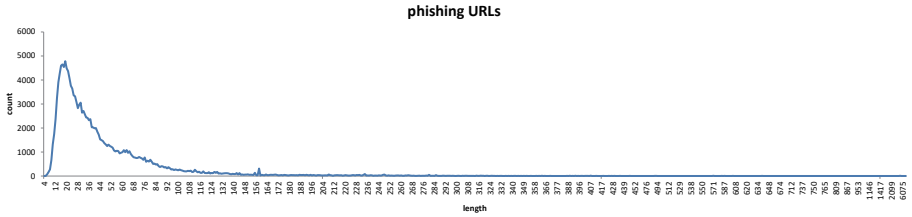

Figure 13. Distribution of URL length in D1.

We have proceeded five experiments on the above mentioned datasets. In experiment 1, feature groups FG1∼FG4 are extracted from D1 and evaluated using different models. In experiment 2, both FG1 and FG2 features are evaluated using different models with regard to D3. Experiment 3 extracts and evaluates features FG2 using different deep leaning models on D2. In experiment 4, the proposed model is evaluated on D1, D2, and D3. Experiment 5 is about comparing our work with existing works. A summary of the five experiments is shown in Table 4.

Table 4. Summary of the five experiments.

The Experiment	The Purpose of the Experiment	The Outputs of the Experiment
Experiment 1	A comprehensive comparison to expose the best classifiers suitable for URL group features	The superior URL group features with respect to classifiers
Experiment 2	Evaluate the superior feature groups with respect to classifiers in experiment 1 using different dataset	The best feature group with respect to classifier
Experiment 3	Evaluate the best feature group with respect to classifier in experiment 2, with different neural network models using the same feature group	The best model as a proposed model
Experiment 4	Evaluate the best model (the proposed model) in experiment 3 using different benchmark datasets	Evaluation of the proposed model based on different statistics (i.e., accuracy, precision, recall, F1-Score, AUC value, training time, and test time)
Experiment 5	Comparing the proposed model with existing baselines in order to evaluate the performance of our model	Evaluation values of our model and other models based on different statistical metrics used in the papers

To measure the effectiveness of our model, we used different statistics metrics such as sensitivity or recall, accuracy, precision, F1_score, AUC and they are calculated as follows:

$$Accuracy = (TP + TN)/(TP + TN + FP + FN) \tag{8}$$

$$Precision = (TP)/(TP + FP) \tag{9}$$

$$Recall = (TP)/(TP + FN) \tag{10}$$

$$F\_Measure = 2 * precision * recall/(precision + recall) \tag{11}$$

where TP is the true positive that denotes the number of URLs which are phishing and correctly classified, TN is true negative that denotes the number of benign URLs recognized as benign ones, FP is false positive that denotes the number of benign URLs which are wrongly classified as phishing, and FN is the false negative that denotes the number of phishing URLs identified as benign ones.

The receiver operating characteristic (ROC) arch and AUC are usually used to evaluate the merits of a binary classifier. The horizontal coordinate of the ROC arch is FPR, which denotes the possibility that the normal URL of the website is incorrectly marked as a phishing; the ordinate is TPR, which

denotes the possibility that the phishing URL of website is correctly classified as a phishing website. Their definitions are as follows:

$$FPR = (FP)/(FP + TN) \tag{12}$$

$$TPR = (TP)/(FP + TN) \tag{13}$$

4.2. Evaluation of FG1, FG2, FG3, and FG4 on D1 with Different Classifiers

In this experiment, we compared different classifiers such as Multinomial Naïve Bayes (MNB), Logistic Regression (LR), Gaussian Naïve Bayes (GNB), random forest (RF), XGBoost (XGB), deep neural network (DNN), and convolutional neural network (CNN) using all URL group features FG1∼FG4 on D1. The main purpose of this experiment was to expose the best classifiers suitable for URL group features. The comparison results are shown in Table 5. From the results, it is observed that MNB, LR, and GNB classifiers have good accuracy, precision, recall, F-Score, and AUC with FG1 and FG3, whereas MNB, GNB classifiers have low accuracy, precision, F-Score, and AUC with FG2 due to the independent predictors and the normal features distribution that assumed by Naïve Bayes classifier. RF and XGB classifiers are a type of ensemble classifiers, that convert weak learners to strong ones, and suitable for all feature groups, thus they have perfect accuracy, recall, F-Score, and AUC with FG1, FG2, and FG3. DNN has high accuracy, precision, F-Score, and AUC with regard to FG1, high recall with FG1∼FG3, good accuracy, precision, F-Score, and AUC with FG2 and FG3. Since convolutional networks use embedding and convolution layers and deal with sequential data, CNN is suitable for FG2 and outperforms the others with an accuracy of 95.02 %, F-Score of 95.13%, and AUC of 95.04%. With regard to FG4, all classifiers have normal accuracy, and the reason behind less accuracy is due to features that are not efficient enough to clearly distinguish between phishing and benign URLs, as phishers use new techniques to prepare deceptive URLs that are very similar to the benign URLs.

Table 5. The results of the classification models on D1.

Model	Features	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC (%)
Multinomial Naïve Bayes	FG1	87.58	89.15	89.25	87.64	87.59
	FG2	55.45	52.97	95.14	68.05	56.19
	FG3	77.06	72.87	85.43	78.48	77.62
	FG4	53.37	52.83	70.89	60.54	53.22
Logistic Regression	FG1	92.67	88.93	97.24	92.89	92.73
	FG2	73.34	72.66	74.01	73.33	73.35
	FG3	88.18	83.41	94.94	88.80	88.27
	FG4	73.47	77.91	66.18	71.57	73.54
Gaussian Naïve	FG1	89.15	95.52	95.49	89.59	89.66
	FG2	55.50	52.80	96.76	68.32	55.83
	FG3	75.71	67.31	99.56	80.16	76.09
	FG4	62.36	94.41	26.98	41.97	62.67
Random Forest	FG1	92.04	87.71	97.56	92.37	92.10
	FG2	91.38	88.70	94.67	91.59	91.40
	FG3	91.10	86.03	97.96	91.61	91.15
	FG4	74.01	78.47	66.81	72.17	74.07
eXtreme Gradient Boosting	FG1	91.34	86.26	98.18	91.84	91.40
	FG2	93.08	88.72	98.62	93.40	93.13
	FG3	90.78	86.03	97.21	91.28	90.84
	FG4	73.98	78.38	66.86	72.16	74.05
Deep Neural Networks	FG1	93.75	91.11	96.85	93.88	93.77
	FG2	85.93	78.42	98.68	87.39	86.08
	FG3	88.04	80.61	99.92	89.23	88.14
	FG4	73.92	78.24	66.93	72.14	73.99
Convolutional Neural Networks	FG1	51.29	50.47	99.84	67.05	51.93
	FG2	95.02	92.35	98.09	95.13	95.04
	FG3	75.00	74.58	73.66	74.12	74.96
	FG4	73.23	69.62	81.25	74.98	73.32

4.3. Evaluation of FG1 and FG2 on D3 with Different Classifiers

From a previous experiment, it has been observed that FG1 are superior to other features with regard to MNB, LR, GNB, RF, and DNN classifiers, whereas FG2 are superior to other features with respect to XGB and CNN classifiers. Therefore, FG1 are evaluated using MNB, LR, GNB, RF, and DNN classifiers while FG2 are evaluated using XGB and CNN classifiers based on D3. Dataset D3 is used for diversity and performance screening. The accuracy for CNN, DNN, and RF classifiers are 95.41%, 95.24%, and 93.62%, respectively. Results of the experiment are shown in Table 6. Obviously, the accuracy of classifiers obtained in this experiment is better compared to previous experiment.

Table 6. The results of the classification models on D3.

Model	Features	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC (%)
Multinomial Naïve Bayes	FG1	87.44	88.43	70.16	78.24	82.90
Logistic Regression	FG1	91.83	92.66	88.97	86.44	80.99
Gaussian Naïve	FG1	80.43	63.97	89.63	74.66	82.84
Random Forest	FG1	93.62	95.54	84.11	89.46	91.12
eXtreme Gradient Boosting	FG2	92.43	93.61	90.66	92.10	92.38
Deep Neural Networks	FG1	95.24	96.17	93.93	95.04	95.20
Convolutional Neural Networks	FG2	95.41	96.18	94.31	95.23	95.38

4.4. Evaluation of FG2 on D2 with Different Deep Learning Models

In this experiment, FG2 are evaluated on D2 using different deep learning techniques such as recurrent neural networks (RNN), CNN, recurrent convolutional neural networks (RCNN), DNN, and very deep convolutional neural networks (VDCNN) [50]. Table 7 shows the results of different neural network models. From the results, the CNN model has the highest accuracy and best detection performance among all models. It can be seen that the CNN model has higher accuracy of 98.58% on D2 than that of D1 and D3 with respect to FG2. For training, we set the batch size as 128 and the epoch number as 20. The CNN method uses 89 min as training time. VDCNN uses 350 min which is nearly quadruple of CNN model. RNN uses 110 min, RCNN uses 92 min, and DNN uses 4 min. Figures 14–17 show the validation loss and accuracy for the training and testing data of RNN, RCNN, DNN, and VDCNN, respectively.

Table 7. Evaluation of FG2 on D2 with different deep learning models.

Model	AUC	F1 (%)	Precision (%)	Recall (%)	Accuracy (%)
RNN	97.76	97.77	97.91	97.62	97.75
RCNN	98.22	98.18	98.12	98.24	98.21
DNN	81.34	80.84	82.16	79.56	81.36
VDCNN	86.58	84.52	99.77	73.33	86.76
CNN	98.58	98.56	98.55	98.62	98.58

CNN is the proposed model that has been discussed in the methodologies section.

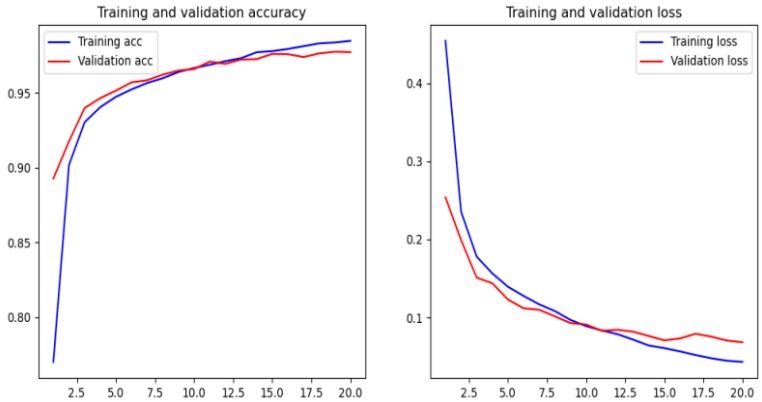

Figure 14. Evaluation of recurrent neural network (RNN) model on D2 using FG2.

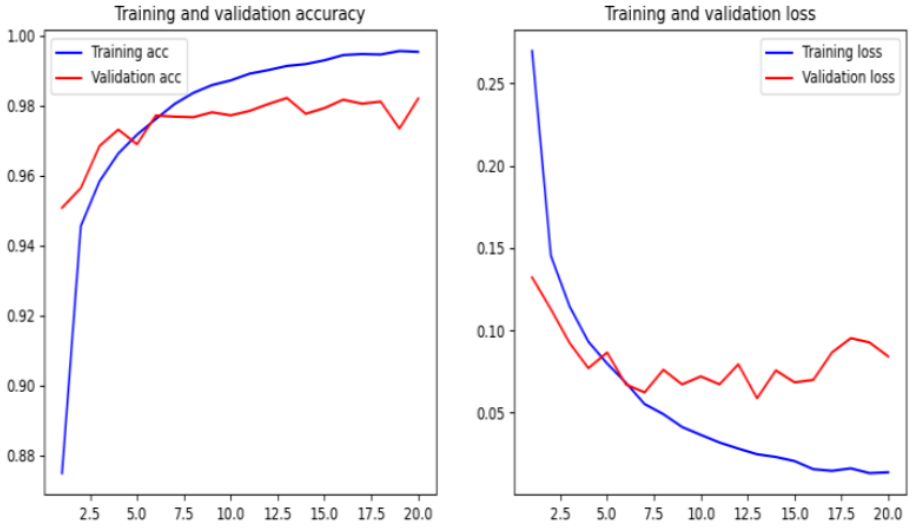

Figure 15. Evaluation of recurrent convolutional neural network (RCNN) model on D2 using FG2.

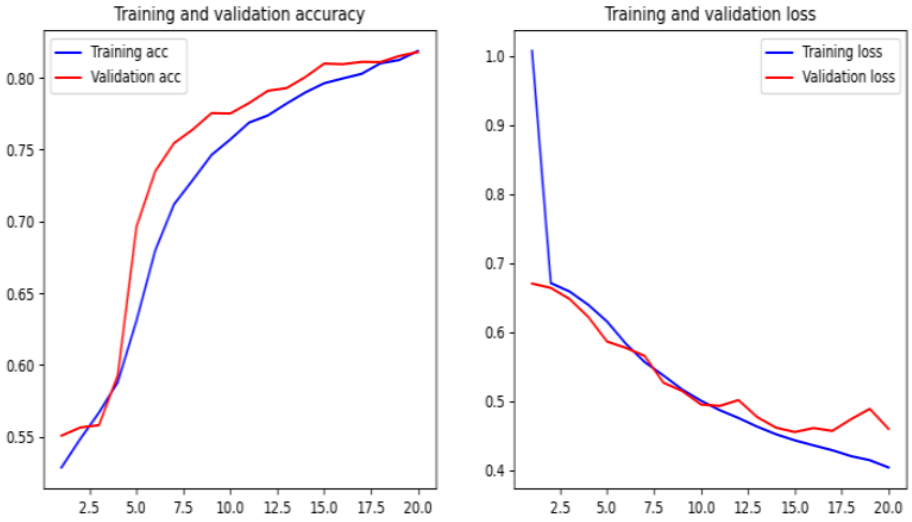

Figure 16. Evaluation of deep neural network (DNN) model on D2 using FG2.

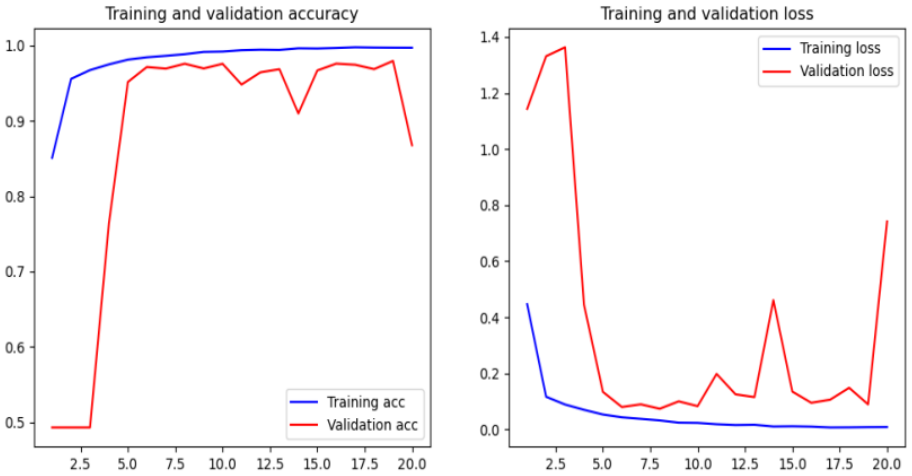

Figure 17. Evaluation of very deep convolutional neural network (VDCNN) model on D2 using FG2.

4.5. Evaluation of the Proposed Model on D2, D3, and D4

Several benchmark datasets that were mentioned earlier are used to evaluate and test the performance of the proposed model in terms of accuracy, precision, recall, F1-Score, AUC value, training time, and test time. The training time indicates the time required to extract features and define optimal parameters on the training set, whereas the test time indicates the time required to detect the classification result for each sample on the test set after the training is completed. The results of this experiment are shown in Table 8. Figures 18–20 show the validation loss and accuracy for the training and testing data on D2, D3, and D4, respectively compared to the proposed model. As shown in the figures, the number of epochs used is 20. Keras callback technique was used to monitor the model during training. Callbacks techniques provide a simple way to save the model automatically when the measure of validation loss or accuracy no longer improving, and they can take action: interrupt training, save a model, load a different weight set, or otherwise alter the state of the model [51].

Table 8. The results of the proposed model on benchmark datasets.

Dataset	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)	AUC (%)	Training Time (s)	Test Time (s)
D2	98.58	98.55	98.62	98.56	98.58	5281.81	32.70
D3	95.46	96.63	94.02	95.30	95.43	6063.63	39.82
D4	95.22	95.01	95.30	95.16	95.22	6027.27	37.48

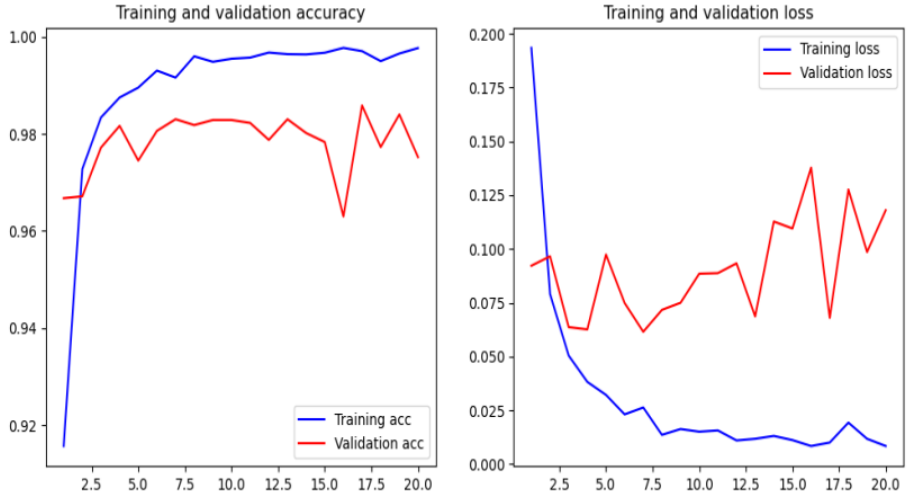

Figure 18. Evaluation of the proposed model on D2.

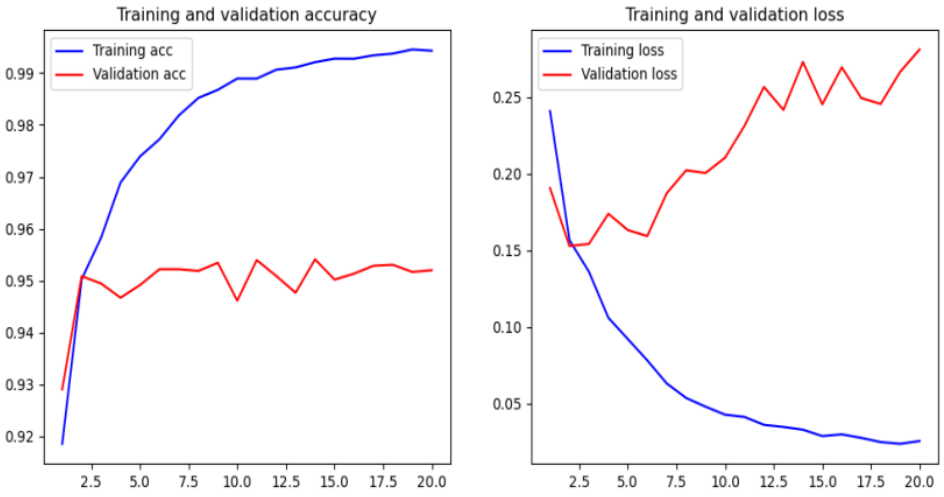

Figure 19. Evaluation of the proposed model on D3.

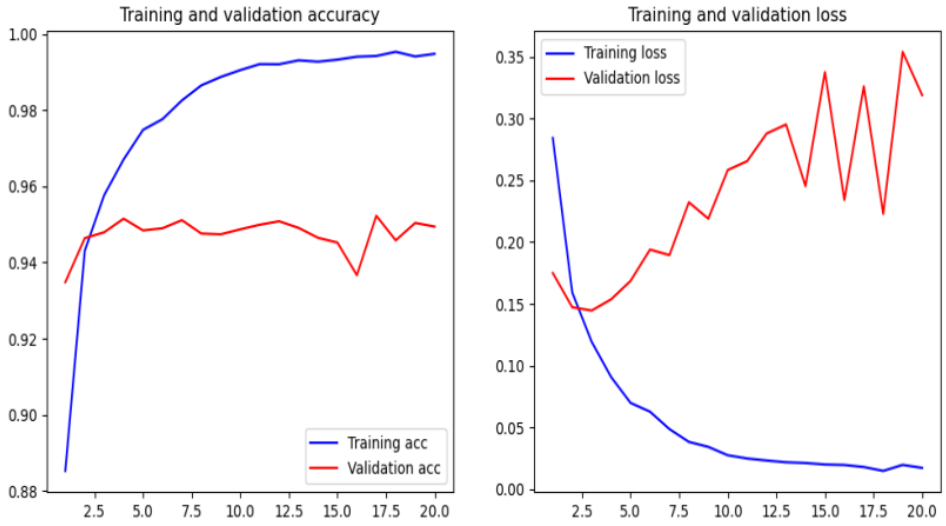

Figure 20. Evaluation of the proposed model on D4.

4.6. Comparison of Proposed Model with Existing Baselines

In this part, we compare the methods of Sahingoz et al. [7], Rao et al. [6], and Le et al. [28] with our proposed method in order to evaluate the effectiveness of our model. Notice that we have implemented Hung Le et al. [28] on dataset D2 whereas for comparison of the proposed method with Sahingoz et al. [7] and Routhu et al. [6], we applied our model on their datasets—D2, D3, and D4. The reason behind the comparison with these works is due to the similarity in phishing detection techniques. These works detect phishing sites based on URLs only without visiting the content of websites.

Rao et al. [6] applied his method to the data of Sahingoz et al. [7] and outperformed Sahingoz et al. [7] with an accuracy of 98.25%, F1 score of 98.23%, and precision of 98.04%. We have applied Le et al. [28] to the data of Sahingoz et al. [7], however the results of Sahingoz et al. [7] and Rao et al. [6] outperformed those of Le et al. [28]. Finally, we applied our method to the data of Sahingoz et al. [7] and our method performed better than the others with an accuracy of 98.58%, precision of 98.55%, F1 score of 98.56%, and sensitivity of 98.62%, which shows the effectiveness of detecting phishing sites over the existing methods. It is also noted that the Sahingoz et al. [7] sensitivity is 0.38% higher than our method because of the use of third-party services. Table 9 illustrates the evaluation values of our method and other methods (Sahingoz et al. [7], Rao et al. [6], Le et al. [28]) based on the four statistical metrics used in the papers.

Table 9. Comparison of the proposed model with existing baseline models on D2.

Metrics (%)	Sahingoz et al.	Rao et al.	Le et al.	Our Model
Precision	97.00	98.04	97.97	98.55
Sensitivity	99.00	98.42	95.32	98.62
F-measure	98.00	98.23	96.62	98.56
Accuracy	97.98	98.25	96.48	98.58

In Table 10, we applied our method to the data of Rao et al. [6] (D3 and D4) and our method outperformed Rao et al. on D4 with an accuracy of 95.22% and F1 score of 95.16%, and on D3 with precision of 96.63%. In order to facilitate comparison, we calculate the three metrics (accuracy, F1 score, and precision) based on the experiment results of Rao et al. It should also be noted that accuracy and F1 score of Rao et al. on D3 and the precision of Rao et al. on D4 are 0.21%, 0.51%, and 0.70%, respectively greater than our method, whereas the accuracy and F1 score of our method on D4 and the precision of our method on D3 are 1.27%, 1%, and 0.32% higher than that of Routhu et al.

Table 10. Comparison of the proposed model with existing baseline models on D3, D4.

Metrics (%)	Rao et al. (D3)	Our Model	Rao et al. (D4)	Our Model
Precision	96.31	96.63	95.71	95.01
F-measure	95.81	95.30	94.16	95.16
Accuracy	95.67	95.46	93.95	95.22

5. Conclusions

In this paper, we have implemented a phishing detection model by using a character level convolutional neural network (CNN). The proposed model based on the features extracted from URL does not need manually designed hand-crafted features, and it is independent of network accessing. This conduct makes the technique suitable at the client side because of its low response time.

We have compared the proposed method by using different machine and deep learning algorithms, such as Naïve Bayes, Logistic Regression, random forest, XGBoost, deep neural networks, recurrent neural networks, recurrent convolutional neural networks, and various types of features as character level TF-IDF features, character embedding features, character level count vectors, and hand-crafted features. The proposed model achieved an accuracy of 95.02% on our dataset and an accuracy of 98.58%, 95.46%, and 95.22% on benchmark datasets.

There are some disadvantages to the model. The main disadvantage is that the training time is rather long, but the trained model is much better than the existing phishing models in terms of accuracy. One of the drawbacks is that the model is not interested if the URL of the website is active or if there is an error. So it is crucial to check the rationality of the URL in anticipation.

Another drawback is that the model may misclassify some of the phishing sites when the URL is short or contains sensitive words such as "login" or "registered", these sensitive words may cause a misclassification of such URLs as phishing websites. Moreover, some URLs for deceptive websites, that are not necessary an imitation of other websites, may not be detected based on a URL string. So our intention in future work is to apply deep learning techniques to feature extraction of webpage code and webpage text.

Author Contributions: Data curation, A.A. and Q.J.; Funding acquisition, Q.J. and Q.Q.; Investigation, Q.J. and Q.Q.; Methodology, A.A. and Q.J.; Project administration, Q.J.; Software, A.A.; Supervision, Q.J.; Validation, Q.Q. and M.H.; Writing—original draft, A.A.; Writing—review & editing, M.H. and J.-P.N. All authors have read and agreed to the published version of the manuscrip.

Funding: This research work is supported by the Key-Area Research and Development Program of Guangdong Province under Grant No. 2019B010137002, the National Natural Science Foundation of China under Grant No. 61902385, and the China Postdoctoral Science Foundation under Grant No. 2020M672892.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Dhamija, R.; Tygar, J.D.; Hearst, M. Why phishing works. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Montreal, QC, Canada, 22–27 April 2006; pp. 581–590.
2. APWG. *Phishing Attack Trends Reports*, 15 May 2019. Available online: https://docs.apwg.org/reports/apwg_trends_report_q1_2019.pdf (accessed on 26 August 2020).
3. Hong, J. The state of phishing attacks. *Commun. ACM* **2012**, *55*, 74–81. [CrossRef]
4. Symantec, Executive Summary. *Internet Security Threat Report*, March 2018. Available online: https://docs.broadcom.com/doc/istr-23-2018-executive-summary-en-aa (accessed on 11 July 2020).
5. Gudkova, D.; Vergelis, M.; Shcherbakova, T.; Demidova, N. Kaspersky Lab: Spam and Phishing in 2017. *Spamand Phishing Report*, 15 February 2017. Available online: https://securelist.com/spam-and-phishing-in-2017/83833/ (accessed on 11 July 2020).
6. Rao, R.S.; Vaishnavi, T.; Pais, A.R. CatchPhish: Detection of phishing websites by inspecting URLs. *J. Ambient. Intell. Humanized Comput.* **2019**, *11*, 813–825. [CrossRef]

7. Sahingoz, O.K.; Buber, E.; Demir, O.; Diri, B. Machine learning based phishing detection from URLs. *Expert Syst. Appl.* **2019**, *117*, 345–357. [CrossRef]
8. Wang, W.; Zhang, F.; Luo, X.; Zhang, S. PDRCNN: Precise Phishing Detection with Recurrent Convolutional Neural Networks. *Secur. Commun. Netw.* **2019**, *2019*. [CrossRef]
9. Yang, P.; Zhao, G.; Zeng, P. Phishing Website Detection Based on Multidimensional Features Driven by Deep Learning. *IEEE Access* **2019**, *7*, 15196–15209. [CrossRef]
10. Yuan, H.; Yang, Z.; Chen, X.; Li, Y.; Liu, W. URL2Vec: URL Modeling with Character Embeddings for Fast and Accurate Phishing Website Detection. In Proceedings of the 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), Melbourne, Australia, 11–13 December 2018.
11. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016.
12. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef] [PubMed]
13. Kim,Y. Convolutional neural networks for sentence classification. *arXiv* **2014**, arXiv:1408.5882.
14. Zhang, X.; Zhao, J.; LeCun, Y. Character-level convolutional networks for text classification. In Proceedings of the Advances in Neural Information Processing Systems 28 (NIPS 2015), Montreal, QC, Canada, 7–12 December 2015.
15. Wang, Y.; Agrawal, R.; Choi, B.Y. Light weight anti-phishing with user whitelisting in a web browser. In Proceedings of the 2008 IEEE Region 5 Conference, Kansas City, MO, USA, 17–20 April 2008.
16. Chou, N.; Ledesma, R.; Teraguchi, Y.; Boneh, D.; Mitchell, J.C. *Client-Side Defense against Web-Based Identity Theft*; Report; Computer Science Department, Stanford University: Stanford, CA, USA, 5 February 2004.
17. Han, W.; Cao, Y.; Bertino, E.; Yong, J. Using automated individual white-list to protect web digital identities. *Expert Syst. Appl.* **2012**, *39*, 11861–11869. [CrossRef]
18. Felegyhazi, M.; Kreibich, C.; Paxson, V. On the potential of proactive domain blacklisting. In Proceedings of the 3rd USENIX Conference on LARGE-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More, Berkeley, CA, USA, 27 April 2010.
19. Rao, R.S.; Pais, A.R. An enhanced blacklist method to detect phishing websites. In Proceedings of the International Conference on Information Systems Security, Porto, Portugal, 19–21 February 2017; Singh, V., Vaidya, J., Eds.; Springer: Cham, Switzerland, 2017.
20. Prakash, P.; Kumar, M.; Kompella, R.R.; Gupta, M. Phishnet: Predictive blacklisting to detect phishing attacks. In Proceedings of the 2010 Proceedings IEEE INFOCOM, San Diego, CA, USA, 14–19 March 2010.
21. Zhang, Y.; Hong, J.I.; Cranor, L.F. Cantina: A content-based approach to detecting phishing websites. In Proceedings of the 16th international conference on World Wide Web, Banff, AB, Canada, 8–12 May 2007; pp. 639–648.
22. Xiang, G.; Hong, J.; Rose, C.P.; Cranor, L. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Trans. Inf. Syst. Secur.* **2011**, *14*. [CrossRef]
23. Rao, R.S.; Ali, S.T. Phishshield: A desktop application to detect phishing webpages through heuristic approach. *Procedia Comput. Sci.* **2015**, *154*, 147–156. [CrossRef]
24. Jain, A.K.; Gupta, B.B. A novel approach to protect against phishing attacks at client side using auto-updated white-list. *EURASIP J. Inf. Secur.* **2016**, *2016*, 9. [CrossRef]
25. Zhang, W.; Ding, Y.X.; Tang, Y.; Zhao, B. Malicious web page detection based on on-line learning algorithm. In Proceedings of the 2011 International Conference on Machine Learning and Cybernetics, Guilin, China, 10–13 July 2011.
26. Moghimi, M.; Varjani, A.Y. New rule-based phishing detection method. *Expert Syst. Appl.* **2016**, *53*, 231–242. [CrossRef]
27. Wei, B.; Hamad, R.A.; Yang, L.; He, X.; Wang, H.; Gao, B.; Woo, W.L. A Deep-Learning-Driven Light-Weight Phishing Detection Sensor. *Sensors* **2019**, *19*, 4258. [CrossRef] [PubMed]
28. Le, H.; Pham, Q.; Sahoo, D.; Hoi, S.C.H. Urlnet: Learning a URL representation with deep learning for malicious URL detection. *arXiv* **2018**, arXiv: 1802.03162.

29. Huang, Y.; Yang, Q.; Qin, J.; Wen, W. Phishing URL Detection via CNN and Attention-Based Hierarchical RNN. In Proceedings of the 2019 18th IEEE International Conference On Trust, Security And Privacy in Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE), Rotorua, New Zealand, 5–8 August 2019.
30. Vargas, J.; Gonzalez, F.A. Classifying Phishing URLs Using Recurrent Neural Networks. In Proceedings of the 2017 APWG Symposium on Electronic Crime Research (eCrime), Scottsdale, AZ, USA, 25–27 April 2017.
31. Jain, A.K.; Gupta, B.B. Two-level authentication approach to protect from phishing attacks in real time. *J. Ambient Intell. Human Comput.* **2017**, *9*, 1783–1796. [CrossRef]
32. Rumelhart, D.E.; Hintont, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
33. Bansal, S. A Comprehensive Guide to Understand and Implement Text Classification in Python. Available online: https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/ (accessed on 1 July 2020).
34. Ramesh, G.; Krishnamurthi, I.; Kumar, K.S.S. An efficacious method for detecting phishing webpages through target domain identification. *Decis. Support Syst.* **2014**, *61*, 12–22. [CrossRef]
35. Lin, M.S.; Chiu, C.Y.; Lee, Y.J.; Pao, H.K. Malicious URL filtering-a big data application. In Proceedings of the 2013 IEEE International Conference on Big Data, Silicon Valley, CA, USA, 6–9 October 2013.
36. Chu, W.; Zhu, B.B.; Xue, F.; Guan, X.; Cai, Z. Protect sensitive sites from phishing attacks using features extractable from inaccessible phishing URLs. In Proceedings of the 2013 IEEE International Conference on Communications (ICC), Budapest, Hungary, 9–13 June 2013.
37. Choi, H.; Zhu, B.B.; Lee, H. Detecting malicious web links and identifying their attack types. In Proceedings of the 2nd USENIX Conference on Web Application Development, Berkeley, CA, USA, 15–16 June 2011.
38. Marchal, S.; Saari, K.; Singh, N.; Asokan, N. Know your phish: novel techniques for detecting phishing sites and their targets. In Proceedings of the 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), Nara, Japan, 27–30 June 2016.
39. Ayres, L.D.G.; Brito, I.V.S.; Souza, R.R.G.E. Using Machine Learning for Automatic Detection of Malicious Brazilian URLs. Available online: https://doi.org/10.5753/sbrc.2019.7416 (accessed on 1 July 2020).
40. Ray, S. Easy Steps to Learn Naive Bayes Algorithm with codes in Python and R. Available online: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/ (accessed on 11 July 2020).
41. Srivastava, T. Building a Logistic Regression Model from Scratch. Available online: https://WWW.ANALYTICSVIDHYA.COM/BLOG/2015/10/BASICS-LOGISTIC-REGRESSION/ (accessed on 11 July 2020).
42. Srivastava, T. Introduction to Random Forest—Simplified. Available online: https://www.analyticsvidhya.com/blog/2014/06/introduction-random-forest-simplified/ (accessed on 11 July 2020).
43. Srivastava, T. How to Use XGBoost Algorithm in R in Easy Steps. 2016. Available online: https://www.analyticsvidhya.com/blog/2016/01/xgboost-algorithm-easy-steps/ (accessed on 11 July 2020).
44. Available online: http://www.alexa.com/topsites (accessed on 1 April 2020).
45. Available online: https://joewein.net/spam/index.htm (accessed on 1 April 2020).
46. Available online: https://www.malwaredomains.com/ (accessed on 1 April 2020).
47. Available online: https://mirror.cedia.org.ec/malwaredomains/ (accessed on 1 April 2020).
48. Available online: http://stuffgate.com/stuff/website/ (accessed on 1 April 2020).
49. Available online: http://www.phishtank.com (accessed on 1 April 2020).
50. Conneau, A.; Schwenk, H.; Cun, Y.L. Very Deep Convolutional Networks for Text Classification. *arXiv* **2017**, arXiv:1606.01781v2.
51. Chollet, F. *Deep Learning with Python*; Manning Publications Co.: Shelter Island, NY, USA, 2018; ISBN 9781617294433.