Information Sciences 631 (2023) 328–345

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

CNN-Fusion: An effective and lightweight phishing detection method based on multi-variant ConvNet

Musarat Hussain [a,b], Chi Cheng [a,b,*], Rui Xu [a], Muhammad Afzal [c,d]

[a] Hubei Key Laboratory of Intelligent Geo-Information Processing, School of Computer Science, China University of Geosciences (Wuhan), China
[b] State Key Laboratory of Integrated Services Networks (Xidian University), China
[c] Department of Software, Sejong University, South Korea
[d] Department of Computing and Data Science, Birmingham City University, UK

ARTICLE INFO

Keywords:
Phishing detection
Deep learning
Convolutional neural network
Phishing attacks
Malicious websites

ABSTRACT

Phishing scams are increasing as the technical skills and costs of phishing attacks diminish, emphasizing the need for rapid, precise, and low-cost prevention measures. Based on a character-level convolutional neural network (CNN), we present CNN-Fusion, an effective and lightweight phishing URL detection method. Our basic idea is to deploy multiple variants of one-layer CNN with various-sized kernels in parallel to extract multi-level features. Observing that differences between phishing and benign URLs might exhibit a strong spatial correlation, we choose SpatialDropout1D, making the model more robust and preventing it from memorizing the training data. To further reduce the probability of errors that may cause by irrelevant or noisy features, we apply a max-over time pooling technique over the feature map to pick only the most important feature. Finally, the model is evaluated using five publicly available datasets containing 1.85 million phishing and benign URLs. Other than that, we assess the model against AI adversarial attacks, known as “Offensive AI.” Compared to existing methods, experiments demonstrate that our approach enjoys advantages in 5 times less training time and much more in memory consumption, achieving an average accuracy above 99% on five different datasets as well as on AI-generated malicious attacks.

1. Introduction

Phishing is a crime that involves both social engineering and technical subterfuge in which Internet users are tricked into disclosing personal or financial account credentials. These sensitive credentials can then be used to perpetrate a variety of frauds such as monetary loss, identity theft, malware installation, and others [2,45]. More and more people are turning to the Internet for a variety of reasons, including social networking, money transfers, e-commerce, and education. As individuals’ time spent connected to the Internet increases, they become increasingly susceptible to various forms of cybercrime. Fraudsters constantly attempt to lure Internet users into their traps to steal their sensitive information by deploying deceptive tactics [32].

* Corresponding author at: Hubei Key Laboratory of Intelligent Geo-Information Processing, School of Computer Science, China University of Geosciences (Wuhan), China.
E-mail address: chengchi@cug.edu.cn (C. Cheng).

https://doi.org/10.1016/j.ins.2023.02.039
Received 10 July 2022; Received in revised form 6 February 2023; Accepted 9 February 2023

Available online 28 February 2023

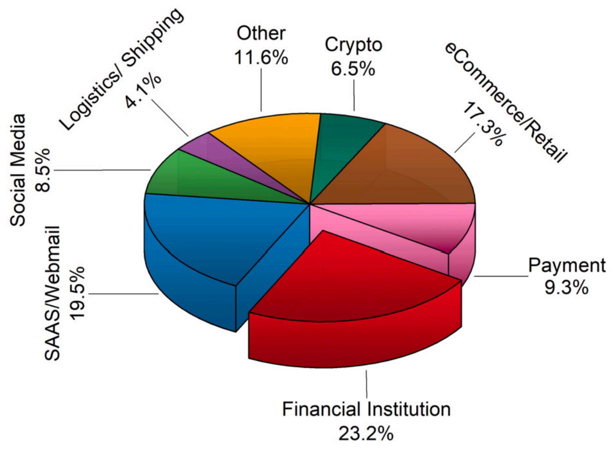


Fig. 1. Most targeted industries for phishing in Q4 2021.

Phishing attacks are on the rise globally. In December 2021, the Anti-Phishing Working Group (APWG)[1] reported 316,747 attempts, the highest monthly attack count in the organization's history. Phishing has become a severe problem due to the tremendous damage it has caused to its targeted industries. Fig. 1 shows the top targeted industries for phishing attacks in the fourth quarter of 2021[1].

It is now possible for anyone, regardless of technological proficiency, to conduct a phishing campaign from anywhere in the world [39]. The majority of phishing attacks attempt to persuade people to visit a fake website in order to collect personal information. Scammers use free webspace to host a fake site or register a new domain [17]. These attacks have become more targeted and effective, causing billions of dollars in economic losses worldwide each year [22]. As a result, it is critical to develop effective ways for identifying malicious URLs quickly and efficiently.

Detecting phishing attacks can be done either reactively or proactively. On the reactive side, we have services like Microsoft SmartScreen Filter,[2] Firefox built-in Phishing and Malware Protection,[3] Google Safe Browsing API,[4] etc. These services expose a blacklist of malicious URLs that can be queried. The blacklists are built using a variety of methods, including manual reporting, honeypots, and web crawling for known phishing characteristics. These methods have some special benefits, such as their high speed and extremely low false-positive rate. For example, when a user reaches a URL that is on a blacklist, their access is blocked by browsers. However, the disadvantage of such reactive methods is that in order for a phishing URL to be blocked, it must first be added to the blacklist. This implies that web users will continue to be vulnerable until the URL is submitted and the blacklist is updated. Furthermore, the blacklist mechanisms eliminate a considerable number of URLs after 5 and 7 days, with none staying after 21 days [7]. Bell and Komisarczuk [7] illustrate that a substantial number of URLs may reappear shortly after being removed, indicating either premature removal or reemergence, thereby significantly reducing the efficiency of blacklists.

In response to these limitations, increasing focus has been given to proactive approaches, i.e., machine learning algorithms to boost the generality of malicious URL detectors [34].

Machine learning-based phishing detection is composed of two steps: first, an appropriate feature representation of a URL is obtained, and then this representation is used to train machine learning-based prediction models. A wide range of features, including lexical features, host-based features, content features, and even context and popularity features, have been investigated [34]. Lexical features are mostly utilized since they have shown good performance and are relatively easy to get [25]. Lexical features explain the lexical aspects obtained from the URL string, such as URL length, number of dots, symbol count, domain length, and so on. Although machine learning-based techniques have exhibited competitive performance, they are constrained by a number of factors. These approaches are incapable of recording semantic or sequential patterns, as well as dealing with unknown features during prediction. Moreover, these techniques require manual extraction of URL features, adding computational and operational overheads [34].

On the other hand, deep learning-based approaches can extract features from raw URLs without the need for human intervention. In response to the pressing need for effective countermeasures, phishing detection has emerged as a thriving area of research during the last few years, with a considerable number of deep learning algorithms being deployed. In particular, convolutional neural network (CNN) [23] and recurrent neural networks (RNNs) [15,3] have shown promising results in detecting phishing attacks.

Recent research has demonstrated that CNNs perform better than RNNs in many different areas of sequence modeling [5]. Le et al. [23] propose URLNet, which employs CNN to learn the URL embedding in a jointly optimized framework using both characters and words from the URL string. Similarly, Huang et al. [20] design a deep learning-based phishing detection approach that extracts character-level features from URL strings using CNN and word-level features using an attention-based hierarchical RNN.

[1] https://apwg.org/trendsreports/.

[2] https://learn.microsoft.com/en-us/windows/security/threat-protection/microsoft-defender-smartscreen/microsoft-defender-smartscreen-overview.

[3] https://support.mozilla.org/en-US/kb/how-does-phishing-and-malware-protection-work#w_how-does-phishing-and-malware-protection-work-in-firefox.

[4] https://developers.google.com/safe-browsing/v4.

However, the promising performance of deep learning models comes at the price of high computational cost. This problem worsens when new URLs (or data) become accessible, and the model needs to be retrained. Taking into account the computational cost and operational overheads of existing deep learning-based phishing detection techniques, we focus on one-layer CNN. In comparison to many sequential layers with a fixed kernel size, multiple one-layer CNNs in parallel with varying kernel sizes enjoy comparative simplicity and good performance in many important natural language processing tasks.

Our main contributions include the following:

1. We propose CNN-Fusion, a character-level CNN based model for malicious URL detection. Our basic idea is to utilize multiple light versions of one-layer CNN in parallel with various-sized kernels to extract features of different sizes from the URL matrix. The features extracted from raw URLs by convolution operations are merged and passed to a fully connected layer. This allows the model to learn an optimal way to integrate these interpretations while processing the input URL at different resolutions or n-grams (groups of n characters) at the same time.
2. We evaluate the impact of various design choices in terms of hyperparameters, regularization, and pooling strategies. Observing that differences between phishing and benign URLs might exhibit a strong spatial correlation, SpatialDropout1D [38] is used. Further, we apply a max-over-time pooling [14] operation over the feature map to pick only the most significant feature, making the model less susceptible to noises. Each outperforms the most commonly used regularization and pooling techniques in phishing detection.
3. We conduct comprehensive experiments on a variety of benchmark datasets and AI-generated adversarial URLs to evaluate the performance of our model. The experimental results show that CNN-Fusion outperforms other methods, with an average accuracy above 99%, even against AI-assisted adversaries. Additionally, the trained model is small and fast, showing its potential to be used on devices with limited memory capacity.

The remainder of the paper is structured as follows: Section 2 presents relevant machine learning and deep learning-based phishing detection efforts. Section 3 describes the proposed methodology. Section 4 contains the findings and discussions, followed by section 5, which covers the closing remarks.

2. Related work

The detection of phishing attacks has been intensively investigated. Despite their high accuracy, some of the methods require complex calculations, which hamper their practical application. The technology for detecting and intercepting phishing threats is continually improving. Numerous investigations are being carried out using machine learning and deep learning techniques in [6,34].

In machine learning-based approaches, a wide range of features is considered. Marchal et al. [28] propose a scalable and language-independent machine learning-based phishing detection technique. They implement the Gradient Boosting algorithm to accurately classify websites as phishing or legitimate. This is achieved through the extraction of 212 features, which serve as inputs to the model. Ghalaty et al. [16] develop a lexical feature-based Random Forest (RF) technique, which obtains 96.79% accuracy. However, manual feature extraction can be time-consuming and complicated [34]. Other researchers in machine learning have proposed representation learning techniques [13,11], which in the context of malicious URL detection, mostly focus on feature selection techniques, i.e., the optimal subset of the given representation to be learned. Chiew et al. [13] develop a two-phase Hybrid Ensemble Feature Selection (HEFS) approach in which a cumulative distribution function gradient is first used to produce primary feature subsets, which are then fed into a data perturbation ensemble to generate the secondary feature subset. However, in addition to the huge volume of URL data, another key challenge in representation learning is the very high dimensional features (often in million or even billion scale). This poses a challenge for training a classification model in practice, and the challenge is far from being solved [34].

To overcome these difficulties, increasing focus has been given to deep learning approaches which learn appropriate features directly from raw data in an end-to-end manner without the need for hand-designed features. Deep learning learns representations at various levels of abstraction by using layers of stacked non-linear projections [24]. It has demonstrated sophisticated performance in various applications, including computer vision, speech recognition, and natural language processing, to mention a few. CNNs, in particular, have become the de facto standard for visual content interpretation in computer vision communities.

Similarly, ConvNets have produced outstanding breakthroughs in the field of text classification in recent years [48,47]. Following the success in text classification, a number of deep learning algorithms have been utilized successfully for malicious URL detection. One key advantage of using deep learning for phishing webpage detection is its ability to learn complex patterns and associations in the data. This is particularly relevant for phishing attacks, which often involve subtle variations on legitimate URLs to deceive users. By training a long short-term memory (LSTM), a deep learning model, Bahnsen et al. [3] achieves a success rate of 97.7%, beating the RF classifier by 5%. This method, however, analyzes the relationship between URL character sequences but does not learn the feature information between local characters. Aljofey et al. [1] propose a character-level CNN that extracts features from raw URLs obtained from numerous fraudulent and harmless websites. Their model achieves an accuracy of 95.02% on a dataset containing 318,642 instances. To obtain diverse representations, Wang et al. [41] propose a phishing detection system based on deep learning that extracts global features from URL strings with bidirectional LSTM and local features with CNN. The extracted features are then combined to classify phishing and non-phishing URLs, resulting in an accuracy of 97%.

Similarly, Yang et al. [44] use CNN to extract local features from URLs, which are then fed into an LSTM network to capture contextual semantics, achieving 98.99% accuracy. Wang et al. [42] propose a malicious URL detection method based on a dynamic convolutional neural network (DCNN). They combine character and word embedding representations and achieve 98% accuracy. In a recent paper, Bozkir et al. [8] propose an adjustable and automated n-gram selection and filtering mechanism, as well as a new neural network architecture that concatenates four-channel information flow through cascading CNN, LSTM, and attention layers. This technique can detect phishing attacks with 98.27% accuracy. Zheng et al. [49] propose highway deep pyramid convolution neural network (HDP-CNN), a deep convolutional network that combines character-level and word-level representation information.

Concurrently, text embedding research using transformers has also produced cutting-edge results in a variety of natural language processing tasks. Wang and Chen [40] study the performance of hybrid transformer and convolutional neural network in detecting phishing URLs and find favorable results across three datasets. Maneriker et al. [26] conduct a thorough analysis of transformer models on the phishing URL detection task. They propose URLTran, which employs a BPE tokenizer and then utilizes a converter to transform the sequence of the token into an embedding vector. In addition, the model can keep a low false positive rate even when subjected to several traditional adversarial black-box phishing attacks. But, it needs to be trained with a massive amount of data and takes a considerable amount of time to train, making it inapplicable in some scenarios.

Most of the above works propose different strategies and combinations of models to defend against phishing attacks. However, the computing power required grows exponentially as the number of neural network layers increases [18], which is an important consideration to keep in mind while developing an effective neural network model for detecting phishing attempts. These models do not have to be complex to produce strong results: for example, Kim [21] suggests a simple one-layer CNN that obtains impressive performance across multiple datasets in sentence classification, which inspired this work. Despite the source of inspiration, our work differs from that of Kim. Kim [21] utilizes both static and dynamic word embedding, whereas we use dynamic character embedding in each variant. Kim [21] employs dropout on the final layer with a constraint on L2-norms of weight vectors while we apply Spatialdropout1D to regularize the convolutional layers and standard dropout to regularize the fully connected layer.

Moreover, in CNN-based approaches, the majority of researchers concentrate on defining parameters such as the number of convolutional layers and kernels, neglecting others like kernel size, pooling, regularization, batch size, number of epochs, or learning rate. The most frequently used metrics to assess the performance of deep learning-based phishing detection methods are accuracy, precision, recall, and F1-score. Fewer researchers pay attention to key metrics like training time, detection time, memory consumption, and floating-point operations (FLOPs). In most cases, a single dataset is used to evaluate a phishing detection technique, which does not adequately capture phishing attacks' dynamic and intelligent nature. What sets phishing apart from other threats is the active and intelligent nature of the attacker, who continuously attempts to bypass any protective measures. In addition to experiments on benchmark datasets, we assess the proposed model's robustness against future AI-generated adversaries by evaluating its performance against malicious AI attacks. From the above perspectives, our approach differs from most of the previous deep learning-based phishing detection solutions.

3. CNN-Fusion

In this section, we explain CNN-Fusion – a deep learning-based phishing detection method.

3.1. Problem formulation

In our approach, we define the problem as a binary classification task to determine whether a given URL is malicious. We consider a set $T = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, where x_i, for $i = 1, \ldots, N$, represents a URL, and $y_i \in \{0, 1\}$ is the corresponding label. When $y_i = 1$, it denotes a phishing URL, while $y_i = 0$ represents a legal one. Then, T can be used to train the model to understand the behavioral pattern of malicious and benign URLs. The first step in the classification procedure is to extract the feature representation of x_i where $\mathbf{x}_i \in \mathbb{R}^K$ is a K-dimensional feature vector. The second step is to learn a prediction function $f : \mathbb{R}^K \rightarrow \mathbb{R}$ which predicts the class assignment for a feature vector $\mathbf{x}$ extracted from any URL instance x. The function's prediction is denoted as $\hat{y}_i = sign(f(x_i))$. The aim is to learn a function that can reduce the total number of errors ($\sum_{i=1}^{N} \mathbb{I}_{\hat{y}i \neq yi}$) in the entire dataset. This is often achieved by minimizing a loss function, and there are many types of loss functions to use.

We use a neural network to learn the URL feature representation and predict whether it is a phishing website.

3.2. Model design

We deploy multiple light versions of one-layer CNN in parallel with varying kernel sizes to search for different features in the input URL at the same time. Fig. 2 depicts the model's overall structure. As demonstrated in [35], the convolutions extract n-gram features from tokens (characters), with varying n-gram lengths needed to comprehend short- and long-span relationships.

CNN-Fusion, in particular, takes a raw URL string as input and applies three-light versions of one-layer CNN in parallel. In simpler words, we apply each convolution to the input data, add a nonlinear function such as Rectified Linear Unit (ReLU), apply SpatialDropout1D followed by max-over-time pooling after each of them, and then concatenate the results and send to a dense layer having 128 units for further interpretation. This enables the input URL to be examined at multiple levels or n-grams simultaneously while the model learns an optimal way to integrate various interpretations.

Since we employ character-level CNN, our objective is to obtain a multi-level representation of a URL sequence of characters that can identify the behavioral patterns of malicious and benign URLs.

Most model pipelines ignore characters that are not in their dictionary and replace them with $< unk >$ tokens. However, the software that calls them may propagate unknown characters from input to output. Although this may help in text understanding

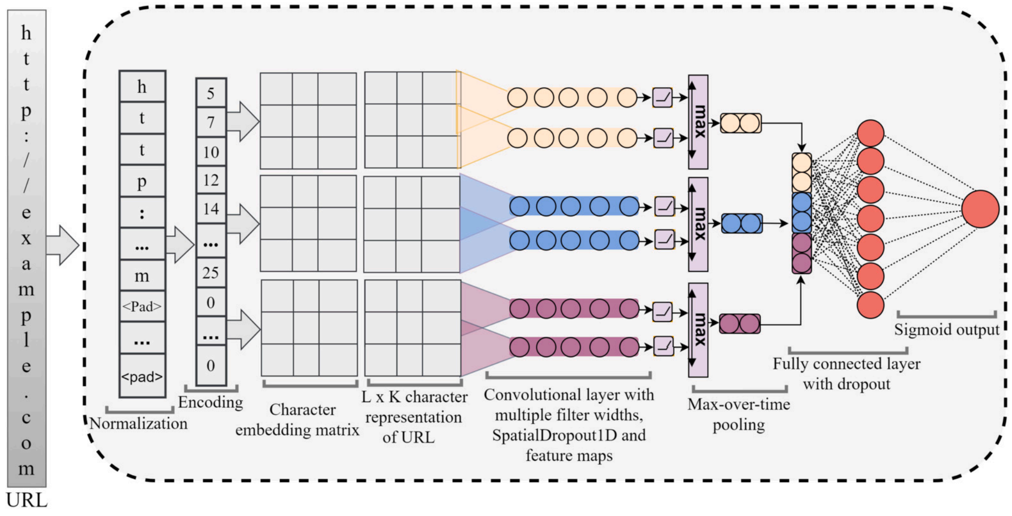

h t t p : / / e x a m p l e . c o m
URL
h t t p : ... m <Pad> ... <pad>
Normalization
5 7 10 12 14 ... 25 0 ... 0
Encoding
Character embedding matrix
L x K character representation of URL
Convolutional layer with multiple filter widths, SpatialDropout1D and feature maps
max
Max-over-time pooling
Fully connected layer with dropout
Sigmoid output

Fig. 2. CNN-Fusion executes three variants of one-layer CNN in parallel, each with a different kernel size to extract multi-level features. The captured features are then combined and interpreted in a dense layer with 128 units. Ultimately, a sigmoid output unit is used for classification.

in general, it creates a surprisingly large attack surface. Taking this into account, first, we create vocabulary by identifying all of the unique alphanumeric and special characters (including non-Latin characters) in the dataset and then encode the characters by assigning a lookup table that takes the integer index of each character in the vocabulary. Since URLs have variable lengths, the maximum number of characters that can be processed in each URL is set to be L. URLs longer than L would get truncated, while URLs shorter than L would be padded.

Each character is embedded into a k-dimensional vector and then the character embedding matrix is learned from random initialization in the end-to-end optimization process. These representations are stored in an embedding matrix $EM \in \mathbb{R}^{M\times K}$, where each row is the vector representation of a character, and M is the size of the vocabulary in a dataset. Using this embedding, each URL x is mapped into a matrix, $x \in \mathbb{R}^{K\times L}$ where we set $K = 16$ and $L = 150$. Using the URL matrix as the training data, we then add a convolutional layer. To explain further operations intuitively, let $x_i \in \mathbb{R}^K$ be the K-dimensional character vector corresponding to the i-th character in the URL string. A URL with a fixed length L is defined as

$$x_{1:L} = x_1 \oplus x_2 \oplus \ldots \oplus x_L, \tag{1}$$

where $\oplus$ is the concatenation operator.

To maximize the learning capability of a deep learning model, we use a nonlinear function ReLU, which is a standard function employed by deep learning models, especially CNN models. To be specific, we need a filter $W \in \mathbb{R}^{K\times h}$ followed by a nonlinear activation function f to generate a new feature, where $\otimes$ is the convolution operation between W and x. The feature c_i, for instance, is produced from a window of characters $x_{i:i+h-1}$ by

$$c_i = f(W \otimes x_{i:i+h-1} + b), \tag{2}$$

where $b \in \mathbb{R}$ is a bias term, and f is a ReLU function. This filter is applied to each possible window of characters in the URL string $(x_{1:h}, x_{2:h+1}, \ldots, x_{L-h+1:L})$ to generate a feature map

$$\mathbf{c} = [c_1, c_2, \ldots, c_{L-h+1}], \tag{3}$$

with $c \in \mathbb{R}^{L-h+1}$. After the convolution operation, we employ a max-over-time pooling operation [14] on the feature map to take the feature with the highest value $\hat{c} = max\{\mathbf{c}\}$ as the filter's feature. In each feature map, we aim to pick the most significant feature with the maximum value. This pooling scheme naturally accommodates the varying lengths of URL strings and discards irrelevant features [21].

3.3. Model regularization

Regularizing neural networks is a vital step to avoid overfitting. To regularize NNs, dropout is the most widely used technique. However, in ConvNets, dropout is generally applied to fully connected layers. The regularization of convolutional layers has received less attention in the literature.

For regularization, we employ SpatialDropout1D [38] on convolutional layers and standard dropout on the penultimate layer. The role of the dropout is to improve generalization performance by preventing activations from becoming strongly correlated, which leads to over-training [19]. In the standard dropout approach, network activations are "dropped-out" (by setting the activation as

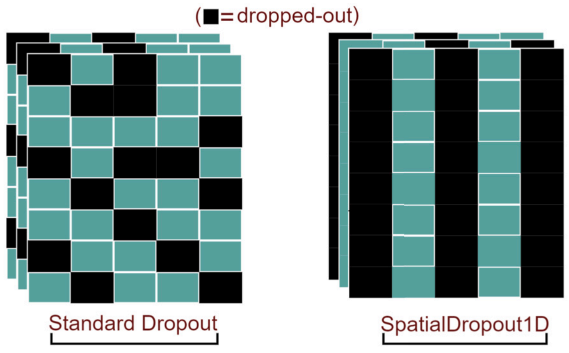


Fig. 3. Illustration of standard Dropout and SpatialDropout1D.

zero for that neuron) during training with independent probability P_{drop}. At test time, all activations are used, but to account for the increase in predicted bias, a gain of 1-P_{drop} is multiplied by the neuron activations.

SpatialDropout1D accomplishes the same function as standard dropout, rather than dropping individual elements, it drops the whole 1D feature map as illustrated in Fig. 3. Since the phishing and benign URLs are quite similar, we believe that they might exhibit a strong spatial correlation, and the feature map activations are also similarly strongly correlated, so in this setting, standard dropout would fail. We also implement Batch Normalization and derive practical observations from our extensive empirical results in subsection 4.5.

3.4. Model optimization

While training a deep learning model, an essential step is to choose an appropriate optimization algorithm. We perform experiments using different optimization algorithms, such as AdaGrad, RMSProp, AdaDelta, and Adam. After many trials and errors, we choose Adam optimizer and based on our experimental findings, it shows superior performance in terms of speed and accuracy parameters compared to other optimizers. Besides, the learning rate is another most important hyperparameter to tune for optimal performance in model training. Initially, we perform experiments by setting the learning rate to be 0.01, 0.005, 0.001, 0.0005, 0.0006, and 0.0007, respectively. After extensive analysis, we finally set the initial learning rate as 0.001, which is the default learning rate for Adam, and decrease it by multiplying with 0.8 after three epochs if the accuracy is not improved. While conventional wisdom suggests that Adam does not require tuning, we observe that altering Adam's initial learning rate and decay scheme offers better accuracy over the default settings. It is not a common practice among others, but Wilson et al. [43] have expressed similar findings using adaptive methods.

The batch size is one of the most critical parameters to consider when training a CNN [31]. We perform experiments with a range of batch sizes, B = 64, 128, 256, 512, and 1024, determining the impact of each value and choosing the optimal value based on its performance and time to converge. Considering the impacts of batch size in terms of time to converge, we determine an optimal batch size value, $B = 512$. We use the cross-entropy loss function, which is also known as the log loss function since the problem is a binary classification task. For a given example, the cross-entropy error between the expected outcome class y and the calculated probability $\hat{y}$ is denoted by:

$$\mathcal{L}(y,\hat{y}) = -(y log\hat{y} + (1-y)log(1-\hat{y})). \tag{4}$$

Additionally, we conduct experiments with varying numbers of training epochs (20, 25, 30, 35, 40, 45, 50, 60, and 100) and find that some datasets (such as DS-4 and DS-5) need fewer training epochs while others require more. After conducting multiple experiments, we determine the optimal value to be 50. Our observations show that accuracy does not improve beyond this value for either of the benchmark datasets. Though, in general, too many epochs may cause the model to overfit the training data, the regularization strategy we employ helps prevent overfitting of datasets that require fewer training epochs. The model's learning and generalization behavior is demonstrated in Fig. 6, section 4.2.

3.5. Model configuration

The overall picture of the model configuration is depicted in Fig. 4. The model receives a raw URL as an input and processes it by three light variants of one-layer CNN.

We embed each character of the ULR into a 16-dimensional vector and set the length of each URL as 150. We use 512 kernels in total with 128, 128, and 256 for each height of 8, 10, and 12 characters, respectively. These heights efficiently capture patterns in groups of 8, 10, and 12 characters in the input URL at the same time. The convolution layer in each variant is regularized by Spatialdropout1D [38]. The dropout rate is set to be 0.4 for DS-1 whereas it varies for smaller training datasets up to 0.6. Further, we employ the max-over-time pooling [14] technique in each variant, which selects the most significant feature with the highest value from the entire feature map. The extracted features are then merged and sent to a fully connected layer regularized by dropout and followed by sigmoid output for classification. The model configuration stays unchanged except for the dropout probability.

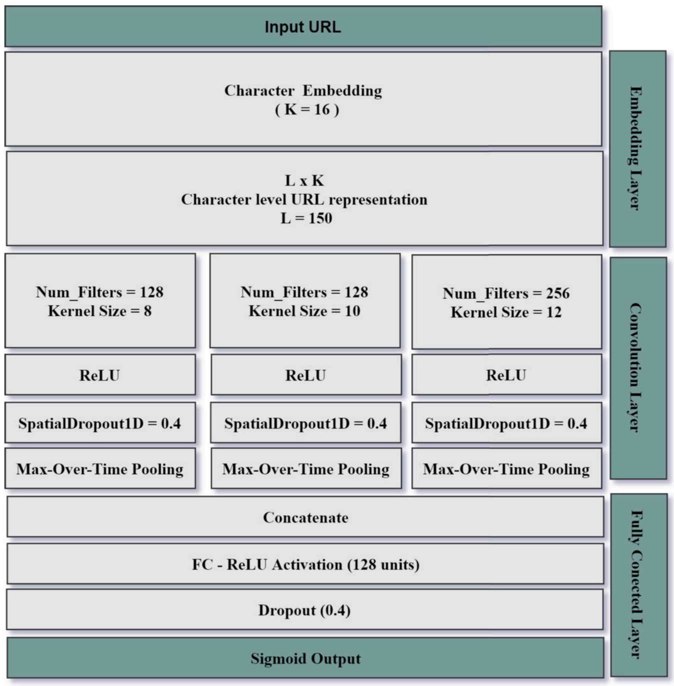


Fig. 4. Configuration of CNN-Fusion.

3.6. *Character embedding*

Phishers employ a variety of URL obfuscation techniques, including typosquatting, bitsquatting, homograph attack, punycode representation, and others. Besides, researchers recently have demonstrated that a deep neural network-based system [4], can be utilized to generate highly sophisticated AI attacks, known as “Offensive AI.” All of these methods are designed to deceive the user by impersonating a different webpage with an address that is very similar to the original one. For example, the goal of a homograph attack is to transform characters into similar graphics. By modifying one character, paypal.com becomes paypaI.com (can be seen in Table 2, DS-4 phishing samples), with the character “l” (small L) substituted by “I” (capital i). We take each of these techniques into account and streamline phishing detection by employing solely character-level CNN and such minor alterations can only be tackled using character embedding.

However, the implementation of CNN in some of the previously explored phishing detection methodologies has involved using random and pre-trained word embedding approaches. The publicly accessible word embeddings are unsuitable for phishing detection tasks as they are trained on human-readable texts. For example, Word2vec uses a big corpus of news as input. Contrastingly, Global Vectors for Word Representation (GloVe) is trained on five corpora ranging from Wikipedia to web data. Unlike these corpora, URLs, on the other hand, are not always so easy to decipher since URLs are made up of meaningless words and can also be in many languages. In such cases, knowledge of words is unnecessary; working with characters is sufficient, hence eliminating the necessity for a word-based feature extractor [47].

In addition, the issue of rare words can be addressed at the character-level embedding, which accounts for most of the dictionary and consumes a significant amount of memory, as more than 90% of the terms in the 5 million training corpus appeared only once in [23]. In terms of computer resources, storing all the words and learning their embeddings during training can be incredibly expensive. Character-level CNN model size remains constant as the number of characters stays fixed, eliminating the risk of missing features during testing that is common in word-based models that expand in size with increasing input. Additionally, URL analysis based on characters makes the model language-agnostic, allowing it to be used with any language.

3.7. *Datasets*

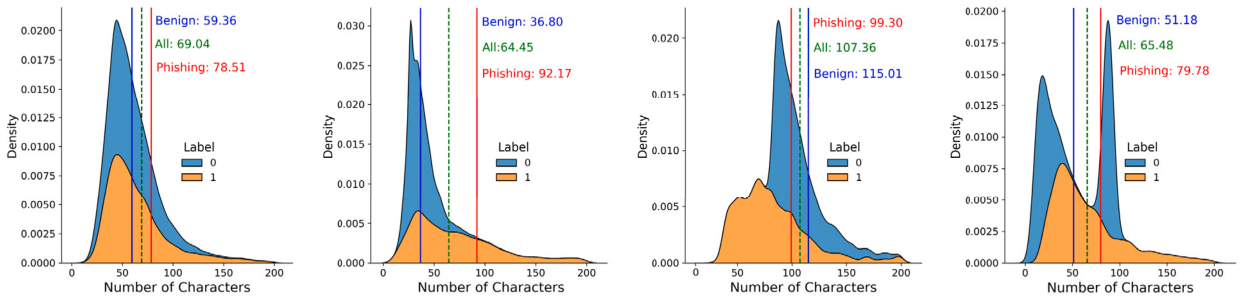
We collect five publicly available real-world URL datasets. The datasets’ sources and other statistics are summarized in Table 1. The first dataset denoted as DS-1, is a large-scale dataset released by [15] on github,[5] containing 1.5 million URLs. The second dataset DS-2 is Ebbu2017, a worldwide known dataset released by [33] on github.[6] The third dataset DS-3 named PhishStorm

[5] https://github.com/vonpower/PhishingDataset.

[6] https://github.com/ebubekirbbr/pdd/tree/master/input.

Table 1
Summary of the datasets' statistics.

Dataset	URL label	Samples	Total Samples	Sources
DS-1	Phishing	759,361	1,559,361	PhishTank
	Benign	800,000		Common Crawl
DS-2	Phishing	37,175	73,575	PhishTank
	Benign	36,400		Yandex
DS-3	Phishing	47,902	95,911	PhishTank
	Benign	48,009		DMOZ
DS-4	Phishing	9000	67,000	OpenPhish
	Benign	35,000		Alexa
	Malware	11,000		DNS-BH
	Spam	12,000		WEBSPAM-UK2007
DS-5	Phishing	26,000	52,000	PhishTank
	Benign	26,000		Alexa



(a) DS-2 (b) DS-3 (c) DS-4 (d) DS-5

Fig. 5. URL length distributions and the mean lengths of phishing, benign, and all URLs.

comes from [27] published by Aalto University.[7] The fourth dataset DS-4, ISCX-URL2016, is released by [25], and can be accessed via Canadian Institute for Cybersecurity website,[8] which contains benign, phishing, malware, and spam URLs and exhibits a 3 : 1 class imbalance. Since we approach a binary classification problem with two classes, we label all non-benign categories of URLs as malicious. The final dataset DS-5 is released by [11] and is available on the author's webpage.[9] All the datasets are divided into the training set, validation set, and test set in a ratio of 8:1:1. Additionally, we compare our results with other machine learning and deep learning-based phishing detection methods applied to the benchmark datasets. A graphical analysis of the URL lengths' statistical distribution is illustrated in Fig. 5

In Fig. 5a and Fig. 5d, it can be observed that the mean length of phishing URLs (measured in number of characters) is lengthier than benign URLs. Similarly, Fig. 5b demonstrates that the mean length of phishing URLs is nearly three times higher than that of legitimate URLs. A lengthy URL increases the likelihood of the user becoming confused. However, in Fig. 5c, the trend is the opposite, with benign URLs having a longer mean length compared to phishing URLs. Since DS-1 has already been preprocessed, we could not visualize it. According to Feng and Yue [15], the mean lengths of phishing, benign, and all URLs are 78.0, 69.7, and 73.8 characters, respectively. Samples of phishing and benign URLs can be seen in Table 2.

In Table 2, it can be observed that the legitimate and malicious URLs are very similar, and it is pretty tricky for human eyes to distinguish the difference between them. Moreover, in DS-4, we can see the alteration in phishing samples. This is to be expected, as the attacker's objective is to fool web users into believing the phishing site is legitimate.

4. Experiments and analysis

In this section, we present the evaluation of the proposed model on benchmark datasets. Additionally, the model is compared to other models developed using machine learning and deep learning approaches and applied to benchmark datasets.

[7] https://research.aalto.fi/en/datasets/phishstorm-phishing-legitimate-url-dataset.
[8] https://www.unb.ca/cic/datasets/url-2016.html.
[9] https://www.lbustio.com/resources.

Table 2
Sample of phishing and benign URLs. Four samples from each dataset are selected with two phishing and two benign URLs.

Dataset	URL Samples	Label
DS-1	https://sicoopresg.sslblindado.com/index.html	1
	https://radcooldeals.com/hollaatme/office365/	1
	https://investir.lesechos.fr/marches/indices/	0
	http://www.asiaartistawards.com/vote/results/	0
DS-2	https://mail.apaypal.co/Secure/PP/Paypal/	1
	update-accounts-services.com	1
	https://www.depositaccounts.com/banks/selfhelp-cu.html	0
	http://www.veracode.com/security/packet-analyzer	0
DS-3	mail.printakid.com/www.online.americanexpress.com/ index.html	1
	financial-advisor-training.com/security/webscr.html	1
	www.taxsites.com/associations2.html	0
	www.financialaccounting.com/financialprof.htm	0
DS-4	http://www.amaz0nesecure.me/https:/www.amazon.com/ap/sign_in/login.phpl/	1
	http://www.lorenzodelfrate.it/verification.paypaI.com/Update/websc.php	1
	http://gizmodo.com/walmart-challenges-amazon-prime-with-its-own-50-delive-1704271709	0
	http://hdfcbank.com/personal/products/investments/distribution-of-financial-products	0
DS-5	https://focusinevents.com/calendar/bankofamerica.com.login/update.html	1
	http://intella-media.ru/cgi/secure1/Alibaba.com/Login.htm	1
	http://techcrunch.com/2015/05/04/case-is-an-insanely-secure-hardware-bitcoin-wallet/	0
	http://ads-verify-manager-notify-support-center-active-recovery.ml/incorrect_email.html	0

4.1. Evaluation metrics

To fully assess the effectiveness of the proposed model, four commonly used measures; Accuracy, Recall, Precision, and F1-score, are employed, which can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$Precision = \frac{TP}{TP + FN} \tag{6}$$

$$Recall = \frac{TP}{TP + FP} \tag{7}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}, \tag{8}$$

where TP and TN represent true positives and negatives, FP and FN denote false positives and negatives, respectively.

In addition, the learning curves are used for each dataset to diagnose the proposed method's learning and generalization behavior.

Since the lifespan of phishing websites is very short, we further evaluate the model's cost efficiency by analyzing both training and detection time, a key factor to consider in establishing a phishing detection method.

4.2. CNN-Fusion performance on benchmark datasets

In this section, we explain the experimental results of our approach. Overall, our strategy outperforms baseline methods across all metrics. The results of the experiments reveal that the presented model performs better and achieves state-of-the-art results across all the datasets. It exhibits consistent stability among benchmark datasets and does not show significant changes when tuning the hyperparameters, indicating its robustness. In each dataset, 80% of the data is used for training and 10% for internal validation and 10% for evaluation. The model is implemented using Keras (Keras Functional API)[10] with a Tensorflow backend.

Apart from the dropout probability, especially for the smaller training datasets, we do no other dataset-specific tuning.

To monitor the model learning performance, we use learning curves to plot variations in learning performance and show how the model gradually optimizes its internal parameters over time. The examination of the model's learning curves during training can assist us in diagnosing issues such as underfitting, overfitting, or optimal fitting and the adequacy of the representation of the training and validation datasets. Fig. 6 demonstrates the learning and generalization behavior of the proposed method on DS-1, DS-2, DS-3, and DS-4, sequentially.

In Fig. 6a, it can be observed that the validation accuracy converges in just 20 epochs, increasing to over 99.5% from epoch 20 onwards. Furthermore, in Fig. 6b, we can see that the plot of training and validation losses lowers to the point of stability with a minimal generalization gap, indicating a good fit model.

[10] https://keras.io/guides/functional_api/.

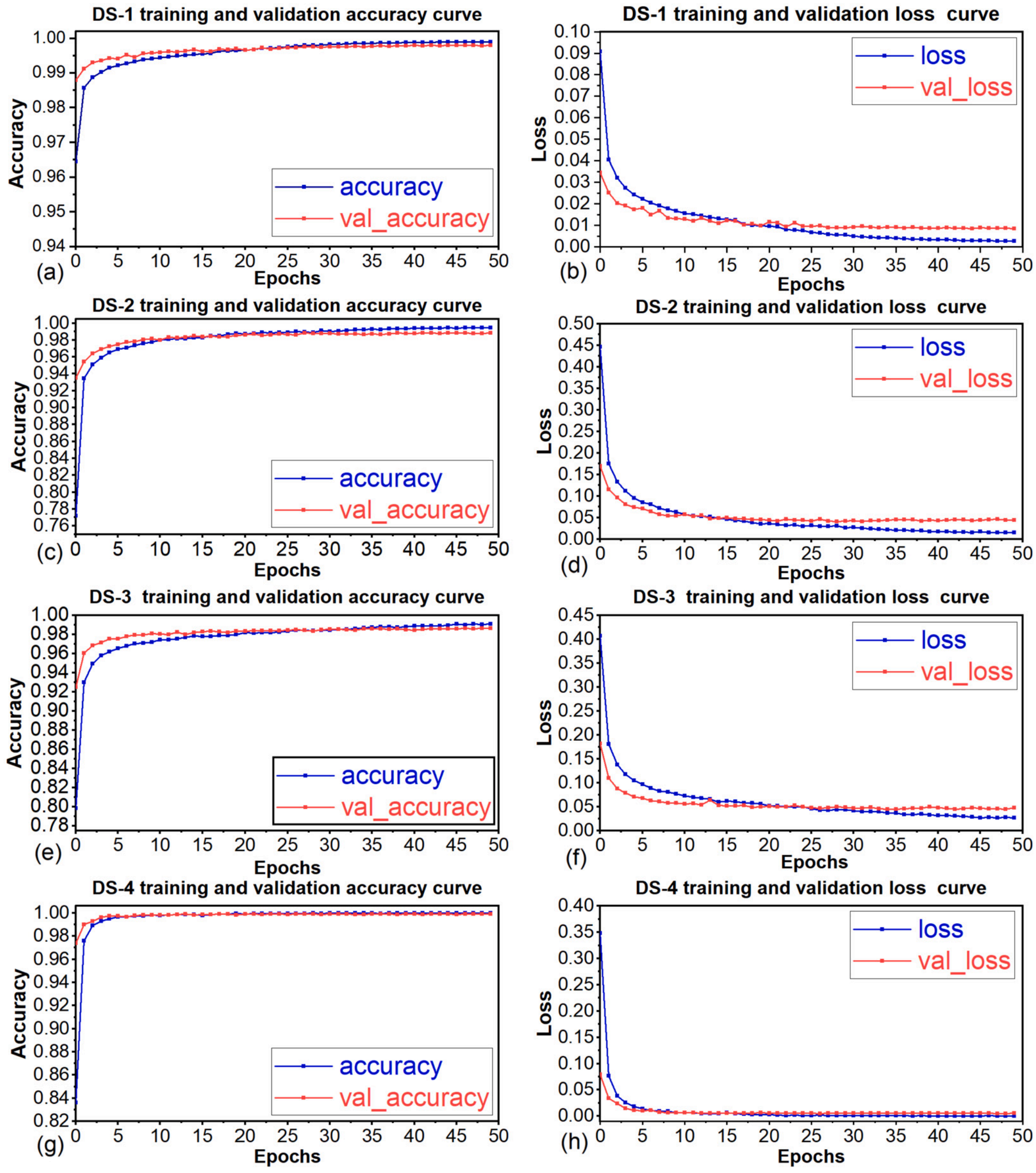


Fig. 6. Accuracy and loss learning curves of the model on benchmark datasets.

Similarly, in Fig. 6c, it can be observed that the validation accuracy exhibits convergence after ten epochs and attains a peak of greater than 98% by the 15th epoch. In Fig. 6d, the minimum generalization gap indicates that the model is neither overfitted nor underfitted. Moreover, as depicted in Fig. 6e, the validation accuracy begins to converge after 20 epochs, achieving a value greater than 98% from epoch 25 onward. Using this dataset, the model produces comparatively lower accuracy than other datasets but higher than other machine learning and deep learning-based methods compared in subsection 4.7. Additionally, as shown in Fig. 6g, the convergence of the validation accuracy is evident in just 5 epochs, reaching a value exceeding 99.8% from epoch 10 onwards. Here, we get the best-fit model among all the datasets. In Fig. 6h, it can further be examined that the plot of training and validation losses lowers to the point of stability with almost no generalization gap indicating that this dataset is relatively easy to learn. The learning behavior of the model on DS-5 is as same as on DS-4, but we do not include it due to space constraints.

The learning curves analysis demonstrates that the proposed model effectively fits the benchmark datasets, and the data split for all datasets is appropriately representative.

In addition, on DS-2, the model produces 38 false positives and 28 false negatives out of 7,358 test samples with 3,718 phishing and 3,640 benign URLs. While utilizing DS-3, it generates only 57 false positives and 65 false negatives out of 9,592 test samples with 4,791 phishing and 4,801 benign URLs. Interestingly, the model produces only one false positive and two false negatives from DS-4 which contains 6,891 test samples with 3,353 phishing and 3537 benign URLs. From DS-5, which has 5200 test samples with 2,600 phishing and 2,600 benign URLs, the model yields only one false positive and three false negatives.

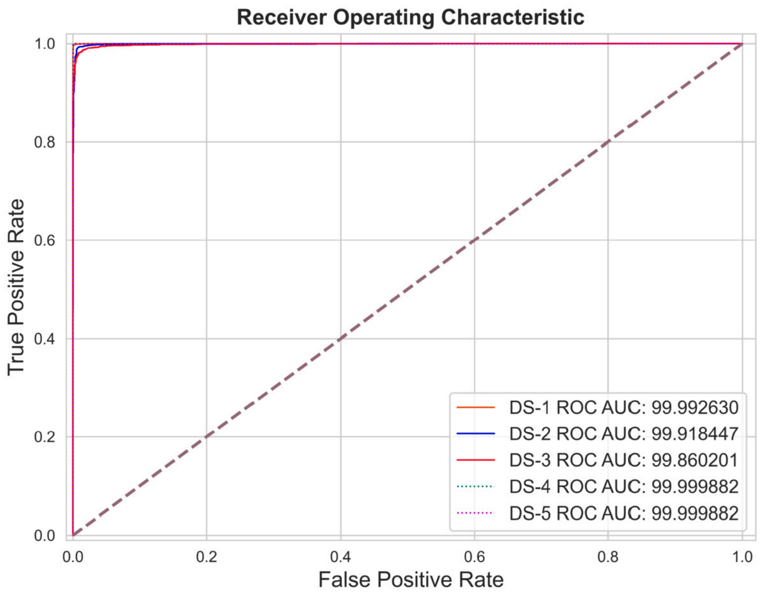


Fig. 7. Area Under ROC Curve. It can be seen that at a very low FPR, the model returns a very high TPR.

Table 3
Effect of convolution's kernel size.

Kernel Size	2	3	4	5	6	7	8	10	12	14
Accuracy %	96.24	97.41	97.47	97.73	97.69	97.93	97.87	98.08	98.01	97.98

Besides, as shown in Fig. 7, the model's AUC (area under the ROC curve) is quite high, indicating that the model has strong classification capabilities with a low false positive rate (FPR) and a high true positive rate (TPR).

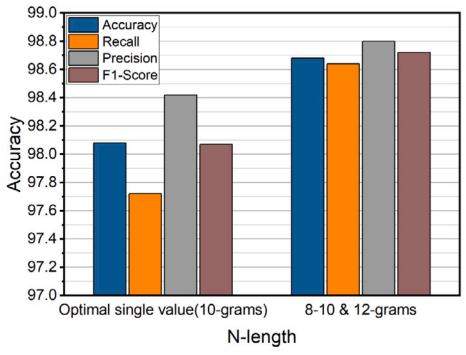
4.3. The effect of convolution's kernel size

In this section, we examine the influence of various n-gram features on model performance. To accomplish this, we analyze the performance of the model in response to the use of different n-gram features, exploring their individual and collective influence on the accuracy of the model. The goal is to find the best combination of n-gram features that can be used to improve the model's performance. Initially, we explore the effect of kernel size using a fixed kernel size in a single variant, with an allocation of 128 feature maps for this particular kernel size. We consider kernel sizes of 2, 3, 4, 5, 6, 7, 8, 10, 12, and 14 and report the average accuracy over 5 replications. We only display the change in accuracy from the baseline kernel sizes (typically ranging from 3 to 6), as we are only interested in the accuracy trend as the kernel size changes. The results can be seen in Table 3.

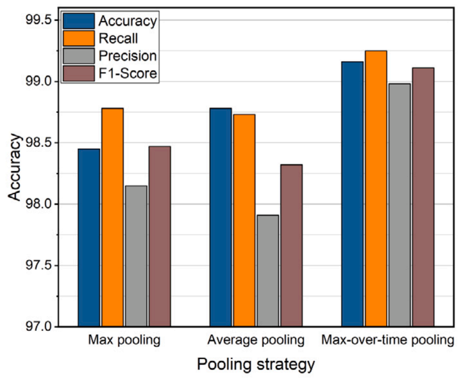
We perform a series of experiments on five different datasets and report the results of DS-3. Since the model's performance on DS-3 is relatively poor compared to other datasets, we perceive during our experiments that the greater the improvement in model performance on DS-3 (and DS-2), the higher the accuracy on the other datasets. Therefore, we prefer to use DS-3 (and DS-2) to assess the impact of various design choices. The results indicate that increasing the kernel size to a certain extent improves the model's performance. From the performance of single kernel size, we can see that the optimal kernel size for DS-3 lies between 7-14, and based on our observations, a similar pattern holds true for other datasets. We then conduct comprehensive experiments with varying kernel sizes in each variant, exploring the kernel size around these values. After conducting a series of experiments on the DS-3, we discover that the model's accuracy steadily improves as the number of features grows and appears to stabilize after reaching 8-grams, 10-grams, and 12-grams in the first, second, and third variants, respectively. Compared to the results obtained with a single kernel size, variable kernel sizes offer better performance. This implies that having different kernel sizes enables the extraction of distinct features from the same input URL simultaneously, resulting in improved accuracy. Results can be examined in Fig. 8a. Moreover, we notice that the model size and training time rise to a certain degree when the number of features increases.

4.4. Effect of different pooling techniques

In this part, we thoroughly examine the influence of various pooling strategies and their impact on model performance using the DS-2 dataset. The purpose is to better understand the role that pooling strategies play in model performance, and to ultimately improve the robustness and accuracy of the model. Through a comprehensive analysis of different pooling techniques, we aim to identify the most effective strategy for phishing detection task.



(a) Comparative findings of kernel size.



(b) Effect of different pooling techniques

Fig. 8. Comparative studies of various kernel sizes and pooling strategies.

In our approach, we perform max-over-time pooling on feature maps globally, resulting in a feature vector of length 1 for each filter. According to our research, max pooling and average pooling are the most commonly used pooling techniques that summarize a feature's most active and average presence, respectively. Max pooling calculates the maximum value, whereas average pooling calculates the average value for each patch on the feature map. Then the pooled features are concatenated to form a single feature vector for the classification layer. We keep the rest of the architecture constant and perform experiments with both techniques. We find that the max-over-time pooling strategy outperforms the other two strategies. The results can be seen in Fig. 8b. Additionally, we observe that max and average pooling result in a high number of trainable parameters (in the millions), leading to increased requirements for disk space, training time, and FLOPs. In comparison, max-over-time pooling generates a comparatively lower number of trainable parameters (around 160k), with a disk space requirement of only about 2 MB.

Furthermore, we explore the global average pooling technique available in Keras pooling layers[11] and find that max-over-time pooling performs better than global average pooling. Our analysis of various pooling strategies reveals that the max-over-time pooling approach consistently exhibits superior performance for the task of phishing detection. This may be attributed to the location of predictive contexts does not matter, and certain n-grams in the URL may individually contain more predictive power than the full URL.

4.5. *Effect of various regularization techniques*

In this section, we study the effectiveness of three regularization techniques for the task of phishing detection using the DS-3 dataset. The techniques evaluated include Dropout, Batch Normalization, and SpatialDropout1D, which are applied to regularize the convolution layers in our model. The comparative results of these techniques are depicted in Fig. 9.

Following extensive experiments, we select SpatialDropout1D for the convolution layers and dropout for the fully connected layer as the final configuration for our model. Instead of individual elements, SpatialDropout1D drops entire 1D feature maps, as illustrated in subsection 3.3 (Fig. 3). If neighboring frames inside feature maps are highly correlated (as is usually the case in early convolution layers), then regular dropout will not regularize the activations and will instead lead to a decrease in effective learning rate [36].

In initial experiments, we observe that using standard dropout and batch normalization on the convolution layer generally increase the training duration but do not prevent over-training. Instead, we apply SpatialDropout1D, where adjacent frames in the dropped-out feature map are either all 0 (dropped-out) or all active. SpatialDropout1D helps to promote independence between feature maps. In addition to this, it also reduces redundant features, helping accelerate the training time. Our findings highlight that this modified dropout technique is more efficient in training time and considerably improves model performance, especially on small training datasets.

Aside from that, the results on benchmark datasets were incredibly consistent when SpatialDropout1D was applied with a similar model configuration. While using standard dropout, we observe comparable variances in the results across datasets when we run the model with the same setup again.

By analyzing the behavior and performance of various techniques used, we can see clear differences in the results. In Fig. 9b, the gap between the training error and validation error is relatively small, suggesting that the model fits the training data well. On the other hand, in Fig. 9d and Fig. 9f, there is a noticeable difference between the training error and validation error, indicating a high variance problem that may affect the model's ability to generalize to new data. The model achieves 98.68% accuracy on DS-3 with the implementation of SpatialDropout1D (Fig. 9a), while standard dropout (Fig. 9c) and batch normalization (Fig. 9e) with a similar model setup and dataset obtain 95.68% and 98.02% accuracy, respectively. The findings of our experiments demonstrate that a well-chosen regularization technique can substantially improve the model's effectiveness and robustness.

[11] https://keras.io/api/layers/pooling_layers/global_average_pooling1d/.

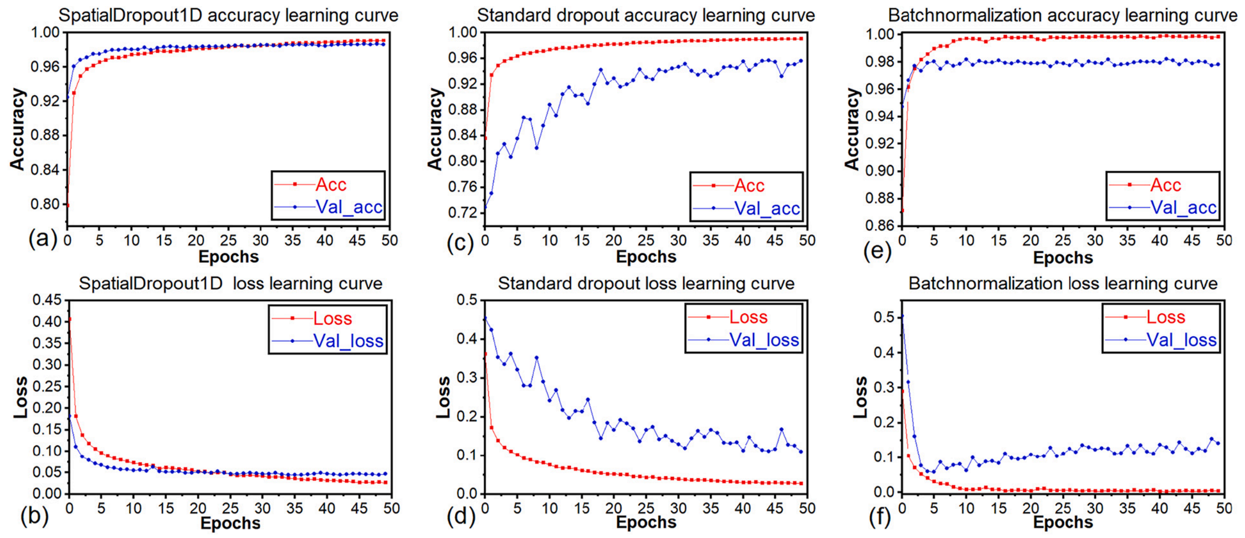


Fig. 9. Effect of SpatialDropout1D, standard Dropout, and Batch normalization.

Table 4
Comparison with URLNet using three benchmark datasets.

Dataset DS-2				
	Accuracy	Precision	Recall	F1-Score
URLNet	97.92%	98.46%	97.39%	97.92%
CNN-Fusion	99.16%	98.98%	99.25%	99.11%
Dataset DS-3				
URLNet	97.61%	96.85%	98.23%	97.53%
CNN-Fusion	98.68%	98.64%	98.81%	98.72%
Dataset DS-4				
URLNet	99.64%	99.84%	99.63%	99.42%
CNN-Fusion	99.96%	99.94%	99.97%	99.96%

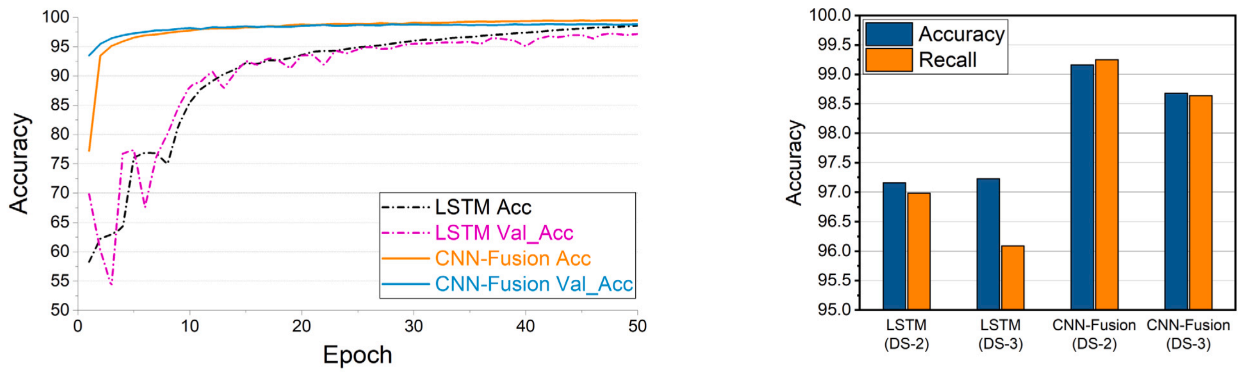
Additionally, we use URLNet as a baseline model to assess CNN- Fusion's performance. The URLNet model [23] is a character-level CNN that is proposed to identify potentially harmful URLs. We obtain the URLNet code from the authors' online repository[12] and execute it on three datasets with default parameters except for the number of training epochs. We compare URLNet with our model in terms of performance and time metrics. Performance comparisons can be seen in Table 4. It can be observed that CNN-Fusion is more precise than URLNet. The impressive effectiveness of this relatively simple CNN architecture suggests that it could be used as an alternative to well-known baseline models.

Similarly, we implement Bahnsen et al. [3] LSTM network, utilizing URL data at the character level, using the default hyperparameter specifications outlined in their study. We only use the DS-2 and DS-3 datasets to train and test their model. The results are provided in Fig. 10. In Fig. 10b, it can be seen that our method outperforms the Bahnsen et al. [3] model. Moreover, in Fig. 10a, a significant difference can be seen in the training of both approaches. The faster convergence shown in Fig. 10a indicates that CNN is easier to train than an LSTM model.

Finally, we evaluate the model's consumption of training and detection time, as well as its storage requirements, in comparison to URLNet. The comparison takes place on a Xeon E5 machine with 48 GB of memory, Intel E5-2620 v4 CPU @2.10 GHz, and a GeForce GTX 1080 Ti GPU. CNN-Fusion generates a relatively smaller number of trainable parameters (around 160k) and requires only about 2 MB of disk space. Meanwhile, URLNet [23], generates millions of trainable parameters, requiring more space, time, and FLOPs. The experimental results of the model cost can be seen in Table 5.

The comparison of CNN-Fusion and URLNet's training and testing times reveals that CNN-Fusion is faster. The CNN-Fusion model requires 3 minutes and 35 seconds to train a dataset containing 66,217 samples, while it takes only 2 seconds to make predictions on a test dataset containing 7,358 samples. In comparison, URLNet requires 10 minutes and 9 seconds for training and 7 seconds for making predictions using a similar dataset. It is important to note that the memory requirement for URLNet is nearly 47 times that of CNN-Fusion. Besides, we observe that an increase in training data results in a substantial rise in training and testing time

[12] https://github.com/Antimalweb/URLNet.



(a) Training and Validation Accuracy learning curve (b) Comparative findings

Fig. 10. Training and validation accuracy learning curve (a) and performance comparison (b), of both techniques.

Table 5
Experimental results of the model cost on DS-2.

Model	Training Time Minutes	Detection Time Seconds	Memory Consumption MB
URLNet	10 : 09	00 : 07	49.4
CNN-Fusion	03 : 35	00 : 02	1.96

for URLNet, with a nearly five-fold increase compared to CNN-Fusion. The same holds true for space utilization, suggesting that the increase is due to the inclusion of new words in the training data.

Additionally, we evaluate the FLOPs of the proposed model using keras-flops.[13] The results show that the proposed model has a low FLOPs count of 0.0244 G, demonstrating its efficiency and lightweight nature. This is a crucial factor in ensuring the feasibility of deploying the model in real-world settings with limited computational resources. The experimental results of the proposed model's cost demonstrate that our method is well-suited for implementation on memory-constrained devices such as mobiles. Deploying deep learning on mobile devices is challenging due to limited computational resources. However, the integration of specialized instructions for machine learning computations in modern mobile chipset technology presents promising opportunities for the efficient deployment of deep learning on mobile devices [29].

4.6. Evaluating CNN-Fusion with AI-generated phishing URLs

The defensive technologies for phishing detection that include machine learning and deep learning show significant improvements in detection, lowering the effectiveness and success rates of the attacks. However, threat actors are always looking for new ways to avoid detection systems and their attacks are becoming more sophisticated. Correspondingly, malicious AI can be used for an offense by sophisticated and motivated adversaries [9]. In light of this, Bahnsen et al. [4] propose "DeepPhish," a model specifically designed to generate AI malicious URLs and act as an adversary. Assuming the role of threat actors, the DeepPhish algorithm takes human-crafted phishing URLs as input and generates new synthetic phishing URLs with the objective of maximizing the effectiveness of the attacks. By training the DeepPhish algorithm for two different threat actors, they were able to increase its effectiveness against an existing AI phishing detection algorithm [3] from 0.69% to 20.9% and from 4.91% to 36.28%, respectively. This suggests that in the near future, AI will be used to carry out highly sophisticated malicious attacks, known as "Offensive AI".

Considering the effectiveness of DeepPhish in evading the existing AI detection system, we retrieve the DeepPhish algorithm from the author's online repository[14] and test our model against AI-generated intelligent attacks. We collect 90,000 phishing URLs from PhishTank,[15] randomly select 20,000 URLs from this corpus, and generate 20,000 more using the DeepPhish algorithm. We obtain full path legitimate URLs from Alexa to complete the dataset and name it as DS-AI. Samples of AI-generated phishing URLs can be seen in Table 6.

We apply the Hold-out method to split the dataset into the training set, validation set, and test set in a ratio of 8:1:1 and perform experiments. Moreover, we conduct experiments by mixing human-crafted and AI-generated phishing URLs. The experimental results demonstrate that our proposed CNN-Fusion model is effective in detecting phishing attacks, even when faced with intelligent adversaries who are constantly attempting to overcome defenses. Fig. 11 summarizes the harmonic mean of precision and recall of human-crafted and AI-generated phishing URLs.

[13] https://pypi.org/project/keras-flops/.
[14] https://github.com/albahnsen/DeepPhish_BlackHat_Demo.
[15] http://data.phishtank.com/data/online-valid.csv.

Table 6
Samples of AI-generated phishing URLs.

http://www.duckdns.org/ferei2.pelu.obginp.yanqncerne
http://www.yandexcloud.net/215f37/t07Dh8ftkney=DPIlCMdh
http://www.blogspot.com/fapfwantiigppicot.htmcirid.top/
http://www.fleek.co/nt/gvogqqxz.duckdns.org/gvtst

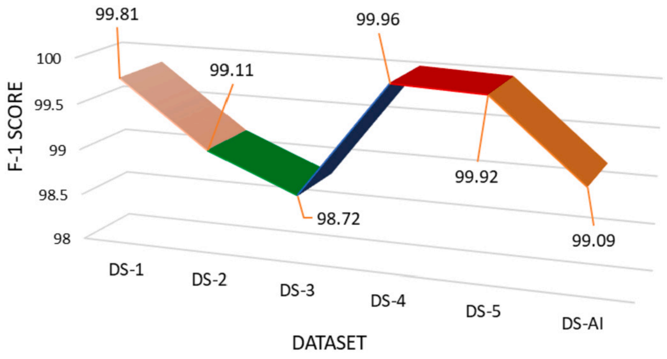
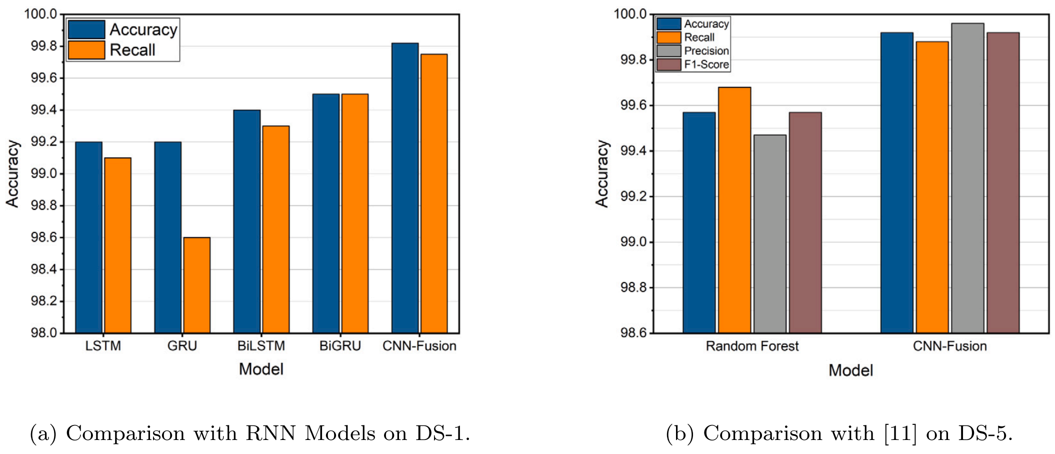


Fig. 11. The harmonic mean of precision and recall of human-crafted and AI-generated phishing URLs.



(a) Comparison with RNN Models on DS-1. (b) Comparison with [11] on DS-5.

Fig. 12. Comparative findings with different phishing detection techniques on benchmark datasets.

4.7. Evaluation of the proposed model in comparison to other existing baselines

This part compares the performance of the proposed strategy to existing methods that utilize machine learning and deep learning techniques with benchmark datasets to detect phishing websites. Initially, experiments were carried out on the vonDataset20180426 dataset, which we denote as DS-1 obtained from [15]. DS-1 is a large-scale dataset consisting of over 1.5 million URLs, with 51% being legitimate and 49% being phishing. The data set is split into training, validation, and testing in the proportions of 8:1:1, respectively. The training and validation sets are used to train and fine-tune the model, while the test set is completely separate and is used to evaluate the final model. The test set has 155,937 samples with 75,775 malicious and 80,162 benign URLs.

Feng and Yue [15] have utilized four variants of RNNs: LSTM, GRU, BiLSTM, and BiGRU, for a similar dataset. More specifically, LSTM and GRU contain 128 cell units, whereas BiLSTM and BiGRU double the number of LSTM and GRU cell units. Comparing the results in Fig. 12a, it can be seen that our method outperforms their RNN models across all the metrics.

Furthermore, the RNN models (i.e., LSTM, GRU, BiLSTM, and BiGRU) have generated 544, 221, 431, and 446 false-positives from the test set, while 697, 1,129, 557, and 392 false negatives were produced, respectively. In comparison to the RNN models' predicted false positives and negatives, CNN-Fusion reports only 93 false positives and 192 false negatives.

Similarly, we obtain the Ebbu2017 dataset, referred to as DS-2, a worldwide known dataset, initially released by [33], and many researchers [12,16,46,30] have adopted it for different machine learning and deep learning-based strategies; results are shown in Table 7. Sahingoz et al. [33] deploy seven machine learning-based classification algorithms and use natural language processing (NLP) features, word vectors, or a mix of the two. The best result (97.98% accuracy) was obtained from Random Forest by employing only NLP-based features.

Table 7
Comparison with other phishing detection approaches on DS-2.

Dataset	Accuracy	Precision	Recall	F1-Score
Chatterjee and Namin [12]	90.10%	86.70%	88.00%	87.30%
Ghalaty et al. [16]	96.78%	90.10%	95.80%	94.45%
Sahingoz et al. [33]	97.98%	97.00%	99.00%	98.00%
PhishTrim [46]	98.34%	98.34%	98.34%	98.30%
DNN-LSTM [30]	98.79%	-%	-%	98.81%
CNN-Fusion	**99.16%**	**98.98%**	**99.25%**	**99.11%**

Table 8
Comparison with Convolutional Autoencoder-based phishing detection approach and the reported models applied to the benchmark datasets by [10]. We assign (a) to their Thresholding based Anomaly score technique and (b) to Threshold Learning with Auxiliary CNN. As indicated in bold, our model overtops the proposed techniques presented by [10] and other reported phishing detection strategies by utilizing the benchmark datasets.

Benchmark Datasets	DS-3		DS-4	
Metrics	Accuracy	Recall	Accuracy	Recall
Character-CNN [48]	90.16%	85.65%	93.63%	89.09%
URLNet [23]	93.95%	88.64%	94.50%	93.90%
Texception [37]	97.10%	92.27%	97.65%	94.42%
Convolutional Autoencoder-based Phishing Detection [10]				
Thresholding based on Anomaly Score (*a*)	95.32%	90.91%	97.34%	93.38%
Threshold Learning with Auxiliary CNN (*b*)	97.32%	93.38%	97.80%	95.90%
CNN-Fusion	**98.68%**	**98.72%**	**99.96%**	**99.94%**

Besides, we obtain an additional dataset from [11] and apply it to our model. Results can be seen in Fig. 12b. Bustio-Martínez et al. [11] propose a feature selection method for efficiently detecting phishing URLs by selecting the best-suited feature set. A feature selection algorithm was used to identify the best representative feature set, and after that, different machine learning algorithms were used for classification. Random Forest, among others, achieves the highest accuracy on the specified feature set. However, a key problem in feature selection techniques is the very high-dimensional features (often in millions or even billion scales) and offer a significant barrier in practice [34].

Moreover, the DS-3 and DS-4 datasets have been utilized by [10] and the authors have made diversified comparisons with the latest deep learning models, including the standard deep learning networks (CNN, LSTM) and their major modifications. Bu and Cho [10] propose a Convolutional Autoencoder-based deep character-level anomaly detection method. They employ three real-world datasets and then compare various deep learning methods. The results of their proposed study and comparative methods applied to the benchmark datasets are provided in Table 8.

After a comprehensive analysis, it can be noted that CNN-Fusion tops all the comparative strategies across all the benchmark datasets developed by other researchers. The proposed method is designed to be flexible and adaptable to changes in the distribution of phishing data characteristics over time. We achieve this by training our model on multiple human-crafted datasets (as well as AI-generated intelligent attacks) covering a range of time periods, which allows the model to be generalized to changes in the way that phishers craft their URLs. Furthermore, the proposed method effectively detects phishing attacks and is designed to be efficient in computation. This makes it suitable for use in real-time situations requiring the evaluation of a large number of URLs. We believe that our approach has the potential to significantly improve the effectiveness of anti-phishing filters in modern web browsers.

Additionally, after conducting extensive experiments, we find that increasing the number of filters can improve performance to a certain extent. We also discover that using a CNN with variable kernel sizes can better identify sequential patterns, and the kernel size can significantly impact performance and should be optimized. We notice that batch size does not affect performance, but a larger batch size can speed up training. While experts generally do not recommend tuning the Adam optimizer, we observe that modifying its initial learning rate and decay scheme can improve model performance. The accuracy of the five benchmark datasets varied, indicating that the model's performance cannot be evaluated based on a single dataset. The model achieved almost 100% accuracy on DS-4 and DS-5, but on DS-3, accuracy decreased by 1.28%. Besides, our observations show that increasing the depth and breadth of the CNN do not improve performance on either dataset, indicating that the complexity of CNNs in phishing detection does not appear to enhance the performance.

5. Conclusion

In this paper, we propose an effective and lightweight character-level CNN that extracts multi-level features from raw URLs without the need for expert knowledge or any third-party services to identify malicious and benign URLs. We explore various design choices through extensive experiments. Our model adopts SpatialDropout1D, which proved to be highly effective in regularization, and applies a max-over-time pooling operation, significantly improving its robustness and overall performance.

The experimental results are quite promising on different datasets collected from various sources, as well as against AI-generated adversarial attacks. The model outperforms the baselines and previously proposed machine learning and deep learning-based techniques applied to the benchmark datasets. The consistent findings across the datasets show that effective optimization of character-level CNN removes unwanted complications and the necessity of word and sub-word level information. This implies that it could be utilized as a drop-in substitute for well-established baseline methods. Furthermore, the experimental findings of the model cost demonstrate that the model is faster and lighter and can be subsequently adapted for use on memory-constrained devices.

One limitation of our model is its relatively low accuracy when dealing with short URLs. Thus, designing an efficient and effective phishing detection solution is a long-term need due to the complexity of phishing campaigns.

CRediT authorship contribution statement

Musarat Hussain: Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Chi Cheng:** Conceptualization, Funding acquisition, Methodology, Validation, Writing – original draft, Writing – review & editing. **Rui Xu:** Conceptualization, Funding acquisition, Validation, Writing – review & editing. **Muhammad Afzal:** Methodology, Validation, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

The work presented in this paper was supported in part by the National Natural Science Foundation of China under Grant nos. 62172374 and 61802354, and Open Research Project of The Hubei Key Laboratory of Intelligent Geo-Information Processing (KLIGIP-2022-B06, KLIGIP-2021-B07).

References

[1] A. Aljofey, Q. Jiang, Q. Qu, M. Huang, J.-P. Niyigena, An effective phishing detection model based on character level convolutional neural network from URL, Electronics 9 (9) (2020) 1514.
[2] Z. Alkhalil, C. Hewage, L. Nawaf, I. Khan, Phishing attacks: a recent comprehensive study and a new anatomy, Front. Comput. Sci. 3 (2021) 563060.
[3] A.C. Bahnsen, E.C. Bohorquez, S. Villegas, J. Vargas, F.A. González, Classifying phishing URLs using recurrent neural networks, in: 2017 APWG Symposium on Electronic Crime Research (ECrime), IEEE, 2017, pp. 1–8.
[4] A.C. Bahnsen, I. Torroledo, L.D. Camacho, S. Villegas, Deepphish: simulating malicious AI, in: 2018 APWG Symposium on Electronic Crime Research (ECrime), 2018, pp. 1–8.
[5] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, arXiv preprint, arXiv:1803.01271, https://doi.org/10.48550/ARXIV.1803.01271.
[6] A. Basit, M. Zafar, X. Liu, A.R. Javed, Z. Jalil, K. Kifayat, A comprehensive survey of AI-enabled phishing attacks detection techniques, Telecommun. Syst. 76 (1) (2021) 139–154.
[7] S. Bell, P. Komisarczuk, An analysis of phishing blacklists: Google safe browsing, openphish, and phishtank, in: Proceedings of the Australasian Computer Science Week Multiconference, 2020, pp. 1–11.
[8] A.S. Bozkir, F.C. Dalgic, M. Aydos, GramBeddings: a new neural network for URL based identification of phishing web pages through n-gram embeddings, Comput. Secur. 124 (2023) 102964.
[9] M. Brundage, S. Avin, J. Clark, H. Toner, P. Eckersley, B. Garfinkel, A. Dafoe, P. Scharre, T. Zeitzoff, B. Filar, et al., The malicious use of artificial intelligence: forecasting, prevention, and mitigation, arXiv preprint, arXiv:1802.07228.
[10] S.-J. Bu, S.-B. Cho, Deep character-level anomaly detection based on a convolutional autoencoder for zero-day phishing URL detection, Electronics 10 (12) (2021) 1492.
[11] L. Bustio-Martínez, M.A. Álvarez-Carmona, V. Herrera-Semenets, C. Feregrino-Uribe, R. Cumplido, A lightweight data representation for phishing URLs detection in IoT environments, Inf. Sci. 603 (2022) 42–59.
[12] M. Chatterjee, A.-S. Namin, Detecting phishing websites through deep reinforcement learning, in: 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), vol. 2, IEEE, 2019, pp. 227–232.
[13] K.L. Chiew, C.L. Tan, K. Wong, K.S. Yong, W.K. Tiong, A new hybrid ensemble feature selection framework for machine learning-based phishing detection system, Inf. Sci. 484 (2019) 153–166.
[14] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, J. Mach. Learn. Res. 12 (ARTICLE) (2011) 2493–2537.
[15] T. Feng, C. Yue, Visualizing and interpreting rnn models in url-based phishing detection, in: Proceedings of the 25th ACM Symposium on Access Control Models and Technologies, 2020, pp. 13–24.
[16] N.F. Ghalati, N.F. Ghalaty, J. Barata, Towards the detection of malicious url and domain names using machine learning, in: Doctoral Conference on Computing, Electrical and Industrial Systems, Springer, 2020, pp. 109–117.
[17] B.B. Gupta, A. Tewari, A.K. Jain, D.P. Agrawal, Fighting against phishing attacks: state of the art and future challenges, Neural Comput. Appl. 28 (2017) 3629–3654.
[18] K. He, J. Sun, Convolutional neural networks at constrained time cost, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 5353–5360.

[19] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinov, Improving neural networks by preventing co-adaptation of feature detectors, arXiv preprint, arXiv:1207.0580.
[20] Y. Huang, Q. Yang, J. Qin, W. Wen, Phishing URL detection via CNN and attention-based hierarchical RNN, in: 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), IEEE, 2019, pp. 112–119.
[21] Y. Kim, Convolutional neural networks for sentence classification, https://doi.org/10.48550/ARXIV.1408.5882, 2014.
[22] O. Kovalchuk, M. Shynkaryk, M. Masonkova, Econometric models for estimating the financial effect of cybercrimes, in: 2021 11th International Conference on Advanced Computer Information Technologies (ACIT), IEEE, 2021, pp. 381–384.
[23] H. Le, Q. Pham, D. Sahoo, S.C.H. Hoi, URLNet: learning a URL representation with deep learning for malicious URL detection, arXiv:1802.03162 [abs].
[24] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[25] M.S.I. Mamun, M.A. Rathore, A.H. Lashkari, N. Stakhanova, A.A. Ghorbani, Detecting malicious URLs using lexical analysis, in: International Conference on Network and System Security, Springer, 2016, pp. 467–482.
[26] P. Maneriker, J.W. Stokes, E.G. Lazo, D. Carutasu, F. Tajaddodianfar, A. Gururajan, URLTran: improving phishing URL detection using transformers, in: MILCOM 2021-2021 IEEE Military Communications Conference (MILCOM), IEEE, 2021, pp. 197–204.
[27] S. Marchal, J. François, R. State, T. Engel, PhishStorm: detecting phishing with streaming analytics, IEEE Trans. Netw. Serv. Manag. 11 (4) (2014) 458–471.
[28] S. Marchal, K. Saari, N. Singh, N. Asokan, Know your phish: novel techniques for detecting phishing sites and their targets, in: 2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS), IEEE, 2016, pp. 323–333.
[29] K. Ota, M.S. Dao, V. Mezaris, F.G.D. Natale, Deep learning for mobile multimedia: a survey, ACM Trans. Multimed. Comput. Commun. Appl. 13 (3s) (2017) 1–22.
[30] A. Ozcan, C. Catal, E. Donmez, B. Senturk, A hybrid DNN–LSTM model for detecting phishing URLs, Neural Comput. Appl. (2021) 1–17.
[31] P.M. Radiuk, Impact of training set batch size on the performance of convolutional neural networks for diverse datasets, Information Technology and Management Science, https://doi.org/10.1515/itms-2017-0003.
[32] S. Rathore, P.K. Sharma, V. Loia, Y.-S. Jeong, J.H. Park, Social network security: issues, challenges, threats, and solutions, Inf. Sci. 421 (2017) 43–69.
[33] O.K. Sahingoz, E. Buber, O. Demir, B. Diri, Machine learning based phishing detection from URLs, Expert Syst. Appl. 117 (2019) 345–357.
[34] D. Sahoo, C. Liu, S.C. Hoi, Malicious URL detection using machine learning: a survey, arXiv preprint, arXiv:1701.07179.
[35] H. Schwenk, L. Barrault, A. Conneau, Y. LeCun, Very deep convolutional networks for text classification, https://doi.org/10.48550/ARXIV.1606.01781, 2017.
[36] B. Shu, F. Ren, Y. Bao, Investigating lstm with k-max pooling for text classification, in: 2018 11th International Conference on Intelligent Computation Technology and Automation (ICICTA), IEEE, 2018, pp. 31–34.
[37] F. Tajaddodianfar, J.W. Stokes, A. Gururajan, Texception: a character/word-level deep learning model for phishing URL detection, in: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2020, pp. 2857–2861.
[38] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, C. Bregler, Efficient object localization using convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 648–656.
[39] J. Vargas, A.C. Bahnsen, S. Villegas, D. Ingevaldson, Knowing your enemies: leveraging data analysis to expose phishing patterns against a major US financial institution, in: 2016 APWG Symposium on Electronic Crime Research (eCrime), IEEE, 2016, pp. 1–10.
[40] C. Wang, Y. Chen, TCURL: exploring hybrid transformer and convolutional neural network on phishing URL detection, Knowl.-Based Syst. 258 (2022) 109955.
[41] W. Wang, F. Zhang, X. Luo, S. Zhang, Pdrcnn: precise phishing detection with recurrent convolutional neural networks, Secur. Commun. Netw. (2019).
[42] Z. Wang, S. Li, B. Wang, X. Ren, T. Yang, A malicious URL detection model based on convolutional neural network, in: International Symposium on Security and Privacy in Social Networks and Big Data, Springer, 2020, pp. 34–40.
[43] A.C. Wilson, R. Roelofs, M. Stern, N. Srebro, B. Recht, The marginal value of adaptive gradient methods in machine learning, https://doi.org/ 10.48550/ARXIV.1705.08292, 2017.
[44] P. Yang, G. Zhao, P. Zeng, Phishing website detection based on multidimensional features driven by deep learning, IEEE Access 7 (2019) 15196–15209.
[45] S. Yoo, S. Kim, S. Kim, B.B. Kang, AI-HydRa: advanced hybrid approach using random forest and deep learning for malware classification, Inf. Sci. 546 (2021) 420–435.
[46] L. Zhang, P. Zhang, PhishTrim: fast and adaptive phishing detection based on deep representation learning, in: 2020 IEEE International Conference on Web Services (ICWS), IEEE, 2020, pp. 176–180.
[47] X. Zhang, Y. LeCun, Text understanding from scratch, arXiv:1502.01710 [abs].
[48] X. Zhang, J. Zhao, Y. LeCun, Character-level convolutional networks for text classification, https://doi.org/ 10.48550/arxiv.1509.01626, 2015.
[49] F. Zheng, Q. Yan, V.C. Leung, F.R. Yu, Z. Ming, HDP-CNN: highway deep pyramid convolution neural network combining word-level and character-level representations for phishing website detection, Comput. Secur. 114 (2022) 102584.