URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection

Hung Le, Quang Pham, Doyen Sahoo, Steven C.H. Hoi
School of Information Systems, Singapore Management University
{hungle, hqpham.2017, doyens, chhoi}@smu.edu.sg

ABSTRACT

Malicious URLs host unsolicited content and are used to perpetrate cybercrimes. It is imperative to detect them in a timely manner. Traditionally, this is done through the usage of blacklists, which cannot be exhaustive, and cannot detect newly generated malicious URLs. To address this, recent years have witnessed several efforts to perform Malicious URL Detection using Machine Learning. The most popular and scalable approaches use lexical properties of the URL string by extracting Bag-of-words like features, followed by applying machine learning models such as SVMs. There are also other features designed by experts to improve the prediction performance of the model. These approaches suffer from several limitations: (i) Inability to effectively capture semantic meaning and sequential patterns in URL strings; (ii) Requiring substantial manual feature engineering; and (iii) Inability to handle unseen features and generalize to test data. To address these challenges, we propose URLNet, an end-to-end deep learning framework to learn a nonlinear URL embedding for Malicious URL Detection directly from the URL. Specifically, we apply Convolutional Neural Networks to both characters and words of the URL String to learn the URL embedding in a jointly optimized framework. This approach allows the model to capture several types of semantic information, which was not possible by the existing models. We also propose advanced word-embeddings to solve the problem of too many rare words observed in this task. We conduct extensive experiments on a large-scale dataset and show a significant performance gain over existing methods. We also conduct ablation studies to evaluate the performance of various components of URLNet.

CCS CONCEPTS

• **Security and privacy → Phishing**; • **Computing methodologies → Neural networks**;

KEYWORDS

URLNet, Malicious URL Detection, Deep Learning

ACM Reference Format:
Hung Le, Quang Pham, Doyen Sahoo, Steven C.H. Hoi. 2018. URLNet: Learning a URL Representation with Deep Learning for Malicious URL Detection. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 13 pages.
https://doi.org/10.475/123_4

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

1 INTRODUCTION

Malicious URLs are one of the primary mechanisms to perpetrate cyber crimes. They host unsolicited content and attack unsuspecting users, making them victims of various types of scams (theft of money, identity theft, malware installation, etc.). This has resulted in billions of dollars worth of losses every year [16]. It has thus become imperative to design robust techniques to detect malicious URLs in a timely manner. Traditionally, and most popularly, this detection is done through the usage of blacklisting methods. These are essentially lists of URLs collected by anti-virus groups which are "known" to be malicious. They are often collected through crowd sourcing solutions (e.g. PhishTank [31]). While these methods are fast (requiring a simple database lookup), and are expected to have low False Positive rates, a major shortcoming is that they cannot be completely exhaustive, and in particular they fail against newly generated URLs. This is a severe limitation as new URLs are generated everyday. To address these limitations, there have been several attempts to solve this problem through the use of machine learning [33]. In particular machine learning models offer the ability to generalize their predictions on new unseen URLs.

Malicious URL Detection through machine learning typically comprises two steps: first to obtain an appropriate feature representation from the URL, and second, to use this representation of the URL to train machine learning based prediction models. The first step of obtaining feature representation deals with obtaining useful information about the URL that can be stored in a vector so that machine learning models can be applied to it. Various types of feature have been considered, including lexical features, host-based features, content features, and even context and popularity features [33]. However, the most commonly used features are lexical features, as they have demonstrated good performance and are relatively easy to obtain [2, 26]. Lexical features describe the lexical properties obtained from the URL string. These include statistical properties such as length of the URL, number of dots, etc. In addition, Bag-of-Words like features are often used. Bag-of-Words indicate whether a particular word or string appears in the URL or not. Consequently, every unique word in the training dataset becomes a feature. Using these features, in the second step, prediction models such as SVMs are trained. These models can be viewed as a form of fuzzy blacklists.

While the above approaches have shown successful performance, they suffer from several limitations, particularly in the context of very large scale Malicious URL Detection: (i) *Inability to effectively capture semantic or sequential patterns*: Existing approaches rely on using Bag-of-Words features, which essentially give information about the presence of a word in the URL. They fail to effectively capture the sequence in which words (or characters) appear in the URL String; (ii) *Require substantial manual feature engineering*:

many of these approaches require expert guidance to determine the important features for the task (e.g. which statistical properties of the URL to use, what type of n-gram features would be better, etc.); (iii) *Inability to handle unseen features*: During prediction, test URLs are likely to contain new words that did not exist in the training data. Under these circumstances, the trained models are unable to extract any useful information about the URL from these words. Moreover, the number of unique words in URLs can be extremely large, causing severe memory constraints while training models.

To address the above issues we propose URLNet, a Deep Learning based solution for Malicious URL Detection. Deep Learning [13, 23, 35] uses layers of stacked nonlinear projections in order to learn representations of multiple levels of abstraction. It has demonstrated state of the art performance in many applications (computer vision, speech recognition, natural language processing, etc.). In particular, Convolutional Neural Networks (CNNs) have shown promising performance for text classification in recent years [18, 39]. Following their success, we propose to use CNNs to learn a URL embedding for Malicious URL Detection.

Specifically, URLNet receives a URL string as input and applies CNNs to both characters and words in the URL. For Character-level CNNs we first identify unique characters in the training corpus, and represent each character as a vector. Using this, the entire URL (a sequence of characters) is converted to a matrix representation, on which convolution can be applied. Character CNNs identify important information from certain groups of characters appearing together which could be indicative of maliciousness. For Word-level CNNs, we first identify unique words in the training corpus, delimited by special characters. Using a word-embedding matrix, we obtain a matrix representation of the URL (which in this context, is a sequence of words). Following this convolution can be applied. Word-level CNNs identify useful patterns from certain groups of words appearing together. However, using word-embeddings faces some challenges: (i) it cannot obtain embeddings for new words at test time; and (ii) too many unique words (specifically in malicious URL detection) - resulting in memory constraints while learning word embeddings. To alleviate these, we propose advanced word-embeddings where the word embedding is learned using the character-level information of each word. This also helps recognize subword level information. Both Character-level and Word-level CNNs are jointly optimized to learn the URLNet prediction model.

1D->2D feature

URLNet allows us to alleviate the shortcomings of traditional approaches such that (i) Character and Word CNNs automatically identify and learn the semantic and sequential patterns in which the characters and words appear in the URL; (ii) Expert feature engineering required is reduced, since the CNN automatically learns features to represent the URL, and we do not rely on any other complex or expert features for the learning task; and (iii) The model learns patterns based on both character and word embeddings. Due to the limited number of characters, this character embedding can generalize to new URLs easily. For word-embeddings, even if the test URLs contain new unseen words, the character-based (advanced word) embedding of the words still allows us to obtain representation for these new words. This way URLNet has superior generalization ability compared to existing approaches. We conduct extensive experiments, analysis and ablation studies to show the efficacy of proposed method.

2 MALICIOUS URL DETECTION

2.1 Problem Setting

Our goal is to classify a given URL as malicious or not. We do this by formulating the problem as a binary classification task. Consider a set of T URLs, $\{(\mathbf{u}_1, y_1), \ldots, (\mathbf{u}_T, y_T)\}$, where $\mathbf{u}_t$ for $t = 1, \ldots, T$ represents a URL, and $y_t \in \{-1, +1\}$ denotes the label of the URL, with $y = +1$ being a malicious URL, and $y_t = -1$ being a benign URL. The first step in the classification procedure is to obtain a feature representation $\mathbf{u}_t \rightarrow \mathbf{x}_t$ where $\mathbf{x}_t \in \mathbb{R}^n$ is the $n-$dimensional feature vector representing URL $\mathbf{u}_t$. The next step is to learn a prediction function $\mathbf{f} : \mathbb{R}^n \rightarrow \mathbb{R}$ which is the score predicting the class assignment for a URL instance $\mathbf{x}$. The prediction made by the function is denoted as $\hat{y_t} = \text{sign}(\mathbf{f}(\mathbf{x}_t))$. The aim is to learn a function that can minimize the total number of mistakes ($\sum_{t=1}^{T} \mathbb{I}_{\hat{y_t} \neq y_t}$) in the entire dataset. This is often achieved by minimizing a loss function. Many types of loss functions can be used, which may also include a regularizer term. For Deep Learning, the function $\mathbf{f}$ is represented as a Deep Neural Network, such as a Convolutional Neural Network.

2.2 Lexical Features

Lexical features have often been adopted for the first stage in which the raw URL $\mathbf{u}$ is converted to a feature vector $\mathbf{x}$. A URL is split into component words which are delimited by special characters in the URL. Using the entire training corpus, all unique words in the training dataset are identified to construct a dictionary, and each word w_i becomes a feature. Given M distinct features, each URL $\mathbf{u}_t$ is then mapped to a vector $\mathbf{x}_t \in \mathbb{R}^M$, such that i^{th} element in $\mathbf{x}_t$ is set as 1 if word w_i is present in the URL, and 0 otherwise. In addition to these Bag-of-Words features, other statistical features are commonly used, including the length of URL, lengths of different segments in the URL, number of dots, etc.

Since the number of unique words can be quite large, the feature size of this dataset is also proportionally large, typically more than the number of URLs present in the corresponding training dataset. Additionally, to retrain the model with new data, the feature size grows, and so does the model size. Here, we describe how exactly using these features are responsible for limitations of existing methods. Three major limitations of these features are: (i) Lack of information about the sequence in which the characters or words appear in the URL. Some strategies have been used to exploit sequential information by creating a separate dictionary for every segment of the URL [2, 26, 27]. For example, this can help distinguish between "com" appearing in the Top Level Domain and "com" appearing the Path of URL. Even then, such strategies do not account for the sequence in which the words or characters appear within a specific segment of the URL. Moreover, these methods cannot exploit information from substrings that may appear within a word of the URL; (ii) Inability to obtain information from rare words. Many words in the URL training corpus appear only once, and if training models such as SVMs are used, these features are unable to provide any useful information; and (iii) Inability to interpret new words in test URLs. Since the new words have never appeared in the training data, the models fail to extract useful information about the URLs from these words.

To address these issues, we propose URLNet.

3 URL NET

In this section, we describe the proposed Deep Learning based URLNet for Malicious URL Detection. The entire network can be visualized in Figure 1.

3.1 Deep Learning for Malicious URL Detection

The underlying deep neural network for URLNet is a Convolutional Neural Network (CNN). CNNs have achieved extraordinary success in Computer Vision tasks [14, 21]. Their designs allowed them to automatically learn the salient features in the images from raw pixel values. Eventually their principles were adopted for Natural Language Processing [10, 17, 18, 39], where the CNNs could learn useful structural information in text from raw values of word or character embeddings. In URLNet, the CNNs are used to learn structural information about the URL. Specifically CNNs are applied at both the character-level and word-level. Next, we describe a simple CNN for URL classification.

A URL $\mathbf{u}$ is essentially a sequence of characters or words (delimited by special characters). We aim to obtain its matrix representation $\mathbf{u} \rightarrow \mathbf{x} \in \mathbb{R}^{L \times k}$, such that the instance $\mathbf{x}$ comprises a set of contiguous components $x_i, i = 1, \ldots, L$ in a sequence, where the component can be a character or a word of the URL. Each such component is represented by an embedding such that $x_i \in \mathbb{R}^k$, is a k-dimensional vector.

Usually, this $k-$dimensional representation for a component is an embedding vector extracted from an embedding matrix that is randomly initialized and jointly learned with the rest of the model. In our work, we randomly initialize the embedding matrix, and learn it in the end-to-end optimization. With this notation, an instance with a sequence of L components can be represented as:

$$\mathbf{x} = \mathbf{x}_{1:L} = x_1 \oplus x_2 \oplus \ldots \oplus x_L$$

where $\oplus$ denotes the concatenation operator. For the purpose of parallelization, usually all sequences are padded or truncated to the same length L.

A CNN would convolve over this instance $\mathbf{x} \in \mathbb{R}^{L \times k}$ using a convolutional operator. A convolution operation $\otimes$ of length h consists of convolving a filter $\mathbf{W} \in \mathbb{R}^{k \times h}$ followed by a non-linear activation f to produce a new feature:

$$c_i = f(\mathbf{W} \otimes \mathbf{x}_{i:i+h-1} + b_i)$$

where b_i is the bias. This convolution layer's output applies a filter $\mathbf{W}$ with a nonlinear activation to every $h-$length segment of its input, each of which is separated by a pre-defined stride value. These outputs are then concatenated to produce output $\mathbf{c}$ such that:

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

After the convolution, a pooling step (either max or average pooling) is applied to reduce the feature dimension and to identify the most important features.

By using a filter $\mathbf{W}$ to convolve on every segment of length h, the CNN is able to exploit the temporal relationship of length h in its input. A CNN model typically consists of multiple sets of filters with different lengths (h), and each set consists of multiple filters. These are hyperparameters of the model that need to be set by the user. A convolution followed by a pooling layer comprises a block in this deep neural network. There can be multiple such blocks that can be stacked on top of each other. The pooled features from the final block are concatenated and passed to fully connected layers for the purpose of classification. The network can then be trained by stochastic gradient descent using backpropagation.

URLNet uses multiple CNNs: one for character-level and one for word-level. Next we describe each component of URLNet in detail.

3.2 Character-level CNN for Malicious URL Detection

Here we present the key ideas for building Character level CNNs for Malicious URL Detection. We aim to learn an embedding that captures the properties about the sequence in which the characters appear in a URL. To do this, we first identify all the unique alphanumeric and special characters in the dataset. including delimiters Characters which appear less frequently (e.g. less than 100 times in a corpus of millions of URLs) are replaced with the unknown token denoted by <UNK>. We obtained $M = 96$ unique characters including the <UNK> and <PAD> tokens. We set the length of the sequence $L_1 = 200$ characters. URLs longer than 200 characters would get truncated from the 200^{th} character, and any URLs shorter than 200 would get padded with the <PAD> token till their lengths reached 200.

Each character is embedded into a $k-$dimensional vector. In our work, we choose $k = 32$ for characters. This embedding is randomly initialized and is learnt during training. For ease of implementation, 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. Using this embedding, each URL $\mathbf{u}$ is transformed into a matrix, $\mathbf{u} \rightarrow \mathbf{x} \in \mathbb{R}^{L_1 \times k}$, where $k = 32$ and $L_1 = 200$.

Using the URL matrix (for all the URLs $\mathbf{x}_t \forall t = 1, \ldots, T$) as the training data, we can now add convolutional layers. We use 4 types of Convolutional filters $\mathbf{W} \in \mathbb{R}^{k \times h}$, with $h = 3, 4, 5, 6$ respectively. Thus, temporal patterns in a sequence of characters of lengths 3, 4, 5, 6 are learnt. For each filter size, we use 256 filters. This is followed by a Max-Pooling layer which is followed by a fully connected layer regularized by dropout. The result is concatenated with other branches of the URLNet, finally leading to the output layer.

Apart from the ability to learn structural patterns in the URL String, Character-level CNNs also allow for easily obtaining an embedding for new URLs in the test data, thus not suffering from inability to extract patterns from unseen words (like existing approaches). As the total number of characters is fixed, the model size of the Character-level CNN remains fixed (unlike models based on words - where model size increases with data size). However, Character-level CNN is not able to exploit information from long sequences of components in the URL. It also ignores word boundaries, making it difficult to distinguish special tokens in the data. Further, in scenarios where the malicious URLs try to mimic benign URLs by having a minor modification to one or few words of the URL [9], Character-level CNN may struggle to identify this information. This is because a sequence of characters with similar spellings is likely to obtain a similar output from the convolutional filters. Thus, Character-level CNNs alone are not sufficient to comprehensively obtain structural information from the URL String, and it is necessary to consider word-level information as well.

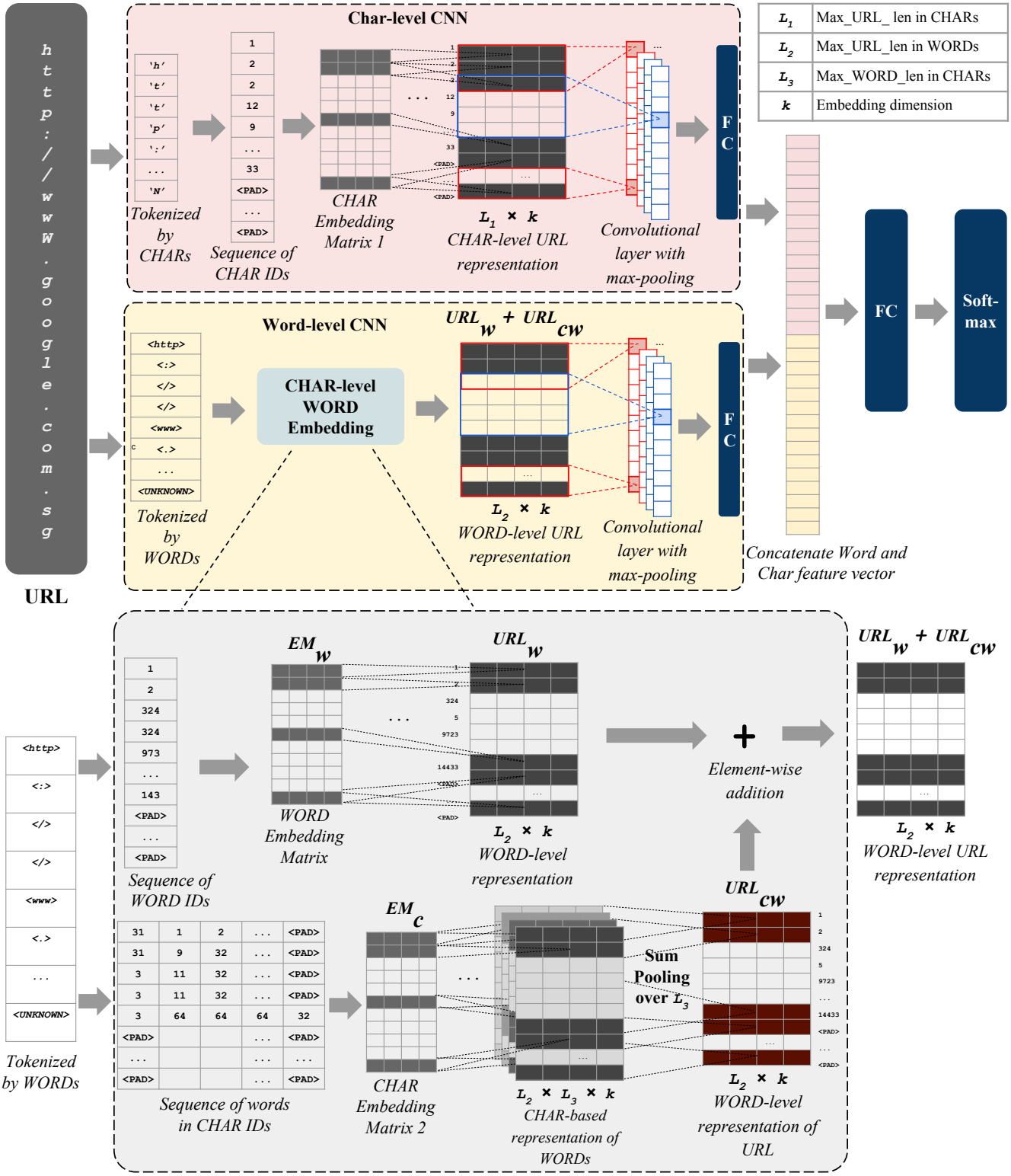

Figure 1: URLNet - Deep Learning for Malicious URL Detection. It comprises two branches with CNNs. The first branch is a Character-level CNN where the character embedding is used to represent the URL. The second is the Word-level CNN where a word-level embedding is used to represent the URL. The word embedding itself is a combination of the individual word's embedding and the character-level embedding of that word.

3.3 Word-level CNN for Malicious URL Detection

Word-level CNNs are similar to Character-level CNNs, except the convolutional operators are applied over words. We present a basic word-level CNN, followed by two advanced methods.

3.3.1 Word-level CNNs. We first identify all the unique words that appear in the training corpus of the URLs. Unlike Character CNNs where the number of unique characters is small and (usually) fixed, the number of unique words depends on the size of the training corpus, as new words can appear in every URL. We identify the unique words using the approaches in Section 2.2, and [2, 26]. All unique words are obtained as a sequence of alphanumeric characters (including '-' and '_'), separated by special characters (e.g. '.', '/', etc.). We also use the <PAD> token as an additional word to make the lengths of the URLs uniform in terms of number of words ($L_2 = 200$). This set of unique words forms a dictionary for the URL training corpus.

In the next step, we obtain the $k-$dimensional vector representation for each word. In our work, we set $k = 32$, i.e., each word is embedded into a 32-dimensional vector. For M unique words, we have to learn an embedding matrix $EM \in \mathbb{R}^{M \times k}$. Using this representation, all the URLs are converted to their respective matrix representation ($L_2 \times k$), on which the CNN is applied. We use the same CNN architecture as in Character CNNs, i.e., we use 4 types of Convolutional filters $\mathbf{W} \in \mathbb{R}^{k \times h}$, with $h = 3, 4, 5, 6$ and for each filter size, we use 256 filters. Here, the aim is to learn temporal properties from a sequence of words of length 3, 4, 5, 6 appearing together. This is followed by a Max-Pooling layer and the fully connected layer regularized by dropout. This output is then concatenated with the other branches of the URLNet.

As the number of words can be extremely large (See Section 4.1.2 for an example), the corresponding Embedding Matrix would become very large, and the model would suffer from memory constraints. As a result, all words that appeared only once in the entire training corpus (also called rare words) were replaced with a single <UNK> token. This would substantially reduce the memory constraints faced by word-based models. During test time, a lot of URLs would contain words that have not been seen during training. Therefore, during conversion of a test URL to matrix form, the unseen words are also replaced with an unknown token <UNK>.

3.3.2 Special Characters as Words. While modeling Word-level CNNs, the words are obtained as a series of alphanumeric characters separated by special characters. This approach does not consider using useful information that can be obtained by evaluating the special characters in the URL String. There are two types of information missed out on: (i) The distribution and types of special characters used; and (ii) The temporal relation of the appearance of words around special characters. To alleviate these issues, we propose to use special characters in the URL Strings as unique words. Conventional approaches in Deep Learning for NLP rarely exploit special characters as words (apart from direct usage in Character-level CNNs). However, we hypothesize that special characters offer significant information gain for Malicious URL Detection because special characters are more frequent and relevant in the context of URLs than normal natural languages. As URL does not follow normal semantic syntax, special characters can play an important feature and should be considered with words.

provides structural information

3.3.3 Improved Word Embedding Using Character-level Word Embedding. The above model does not use the rare words due to memory constraints. Further, it is not able to obtain an effective embedding for new words in test URLs. To address these concerns, we propose to obtain a character-level embedding for each word. In contrast to the previous scenario where the word embedding is obtained directly from the word embedding matrix (which is learnt during training), we obtain the word embedding as a combination of the original word embedding and the embeddings of the individual characters in that word.

Specifically, we maintain 2 Embedding Matrices: one for words ($EM_w \in \mathbb{R}^{L_2 \times k}$), and one for characters ($EM_c \in \mathbb{R}^{L_1 \times k}$). Note that the character embedding matrix here is different from the character embedding matrix used in Character-level CNN. While the Character Embedding Matrix for the Character-level CNN aims to learn character representation based on the full URL, EM_c is more localized, and aims to learn the appropriate character embedding to effectively represent the words. While obtaining the URL Matrix representation, we first get $URL_w \in \mathbb{R}^{L_2 \times k}$ representation based on EM_w. Next, we obtain $L_3 \times k$ matrix representation of each word in a URL (using EM_c), where each word (as obtained in Section 3.3.2) is padded or truncated to be a sequence of $L_3 = 20$ characters. This matrix is summed up to obtain a $1 \times k$ vector embedding for the word. This is applied to all L_2 words in the URL to give us the URL matrix representation $URL_{cw} \in \mathbb{R}^{L_2 \times k}$, which is termed as character-level word embedding of the URL. The final URL matrix representation is simply the sum of these two matrices $URL_w + URL_{cw}$. This entire approach is visualized in URLNet in Figure 1.

During training, the words that appeared only once (rare words) in the entire training data were ignored and converted to a single <UNK> token. However, for each of these words, a character-level embedding was used, thus giving each of the rare words a (mostly) unique representation even during training. In a similar fashion, during test time we are able to obtain a unique word embedding even for the new words not used in training. Thus, the proposed character-level word embedding addresses the issues of memory constraints of word-based models, is able to exploit information from rare words, and also able to obtain a richer representation to capture subword level information.

3.4 Model Configuration

Finally, we present the details of the URLNet model configuration. The overview can be seen in Figure 2. The raw URL input string is processed by 2 branches: a character-level branch and a word-level branch. The character-level branch gives a character-level representation of the URL, while the word-level branch does the same with words. The word-level branch itself is split into word-embedding and character-level word embedding, which are combined to finally give the word-level URL representation. Convolutional operations are applied to both these branches, followed by a fully connected (FC) layer, which is regularized by dropout for both the branches. Then the outputs are concatenated. This is followed by 4 fully-connected layers finally leading to the output classifier. This model is then trained by an optimizer using Backpropagation.

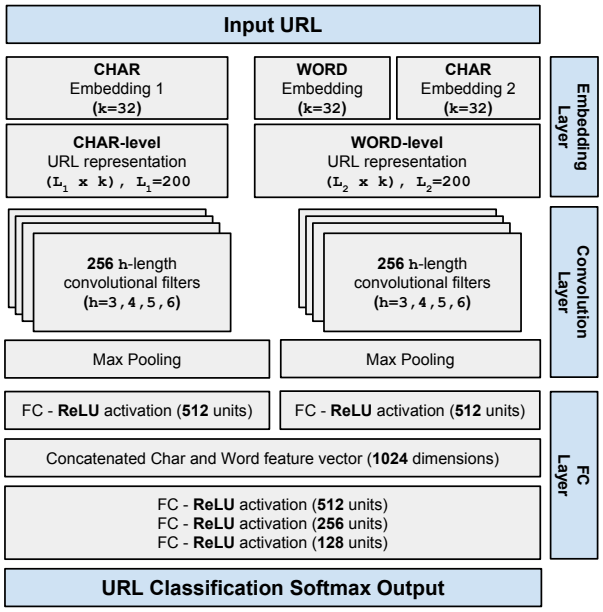

Figure 2: Configuration of URLNet

4 EXPERIMENTS

4.1 Large Scale Dataset

4.1.1 Dataset Collection. We collected a large corpus of labeled URLs from VirusTotal [1]. VirusTotal is an antivirus group whose services are often used to validate whether a given query URL is malicious or not. Given an input URL, VirusTotal scans through 64 different blacklists (e.g. CyberCrime, FraudSense, BitDefender, Google Safebrowsing, etc.), and reports how many of these blacklists contain the input URL. If none of the blacklists contain the given URL, it is assumed that the URL is benign. The higher the number of blacklists the URL is detected by, the higher is our confidence that the given URL is indeed malicious.

We crawled all URLs queried in VirusTotal in the period between May, 2017 to June, 2017, to build our training datatset. From the resulting URLs, we removed any duplicates. We remove the duplicates in order to examine the generalization performance of the models on unseen URLs. However, this makes our benchmark more challenging that traditional (possibly real world) settings. We observed that there were a few dominant domains which occurred very frequently. To reduce any bias, we limited the frequency of any URL domain to be less than 5%. All URLs that did not appear in any blacklist were labelled Benign, and all URLs that appeared in 5 or more blacklists were labelled Malicious. All URLs that appeared in 1, 2, 3, or 4 blacklists were discarded, due to lack of certainty about their true labels. This gave us millions of URLs with roughly 94% Benign and 6% Malicious.

The resulting URLs were sorted according to the timestamp at which they were queried. After sorting, from the first 60% of the URLs, we randomly selected 5 million URLs for training, and from the last 40%, we randomly selected 10 million URLs for testing. The sorting was done to avoid any "look-ahead" bias while training the models. Other details about this corpus can be seen in Table 1.

[1] https://www.virustotal.com/

Table 1: URL Dataset crawled from VirusTotal

	Benign URLs	Malicious URLs	Total
Training	4,683,425	316,575	5,000,000
Testing	9,366,850	633,150	10,000,000
Total	14,050,275	949,725	15,000,000

4.1.2 Feature Extraction. Features are extracted from the raw URLs before training a model. For Character CNNs no feature extraction is required, as the CNN directly operates at the character level. For Word CNN, we extracted lexical features in the form of Bag of Words. For a given training dataset, all unique words are identified and a dictionary is constructed. The number of unique words is denoted by M, and this value varies with the size of the training dataset. This set of features are called *Whole URL BoW*.

We also consider other types of features extracted by security experts for baseline models along the lines of the work in [2, 25, 27]:

- URL Component Tokenization (UCT): In [27], URL is divided into primary domain, path, last path token, and top level domain - and BoW dictionary is constructed for each component. This helps capture some order in terms of sequential information in the URL.
- Position Sensitive & Bigrams (PSB): In [2], in addition to UCT features, specific tokens such as *Domain{1}* and *Path{2}* are extracted to account for sequential information of tokens within the URL primary domain and the URL path. Here *{x}* refers to the zero based token distance from the right most end of a URL part. In addition, token bigrams in primary domain and path are also extracted to further distinguish legitimate and malicious URLs. For example, the feature *Domain{1}{0}* refers to the token bigram consisting of the second last token and the last token in the URL primary domain.
- Character Trigrams: In [25], in addition to UCT features, three character length sliding window is used on the URL domain name to generate character trigram tokens. This is to tackle malicious URLs with subtly modified domain names. We also followed the same feature processing as [25] by only extracting the argument names and discarding the argument values in the URL path.

In addition to the BoW Features, as suggested by the above approaches, a variety of statistical properties of the URL were used, such as the length of the entire URL, the length of the hostname, and the number of dots in the URL. For the baseline [25], we also computed hand designed statistical features such as alphabet entropy measure, character continuity rate, and number rate. We call these statistical features as "Expert Features", as these are hand-designed by experts.

Further details on these Lexical Features can be seen in Table 2. We show the number of words or features for a corpus of 1 million and 5 million URLs. Basic BoW refers to the features obtained following traditional BoW approaches. Advanced BoW refers to features such as bigram and (character) trigram features. As can be seen the total number of lexical features can be very large, and keeps increasing with the data size. Correspondingly, the size of word-based models also keeps increasing.

Table 2: Number of lexical features (words) and expert features and the variation of Feature Size with size of Training Data

	Training Size = 1m				Training Size = 5m			
	#Basic BoW	#Advanced BoW	#Expert Features	Total	#Basic BoW	#Advanced BoW	#Expert Features	Total
Whole URL BoW	1,372,417	N/A	N/A	1,372,417	5,531,997	N/A	N/A	5,531,997
UCT [27]	2,015,183	N/A	3	2,015,186	7,952,252	N/A	3	7,952,255
PSB [2]	2,015,183	4,845,517	17	6,860,717	7,952,252	16,666,144	17	24,618,413
Character Trigrams [25]	1,242,340	54,368	69	1,296,777	4,953,031	58,154	69	5,011,254
Combined	2,015,183	4,899,885	76	6,915,144	7,952,252	16,724,298	76	24,676,626

4.1.3 Word-frequency Distribution. Here, we look at the frequency distribution of the words in dataset. We obtain the whole URL BoW features for 1 million URLs, and plot the percentage of words based on their frequency of occurrence in the entire training data in Figure 3. As can be seen 90% of the words in the training corpus appear only once. This implies that over 90% of the unique words in a URL training corpus are very rare. While rare words make up a majority in the dictionary, storing them requires a lot of memory and is not feasible for large data sets. For example, our training data with 5 million URLs consists of over 5 million unique words. With an embedding size of $k = 32$, these words require more than 150 million parameters for just the embedding matrix (similar to the number of parameters as a VGG-16 network[36]). When scaling to even larger training datasets, storing this matrix becomes infeasible. By ignoring rare words, we are able to resolve the memory issue and make it possible to train word-related models on large data sets. Moreover, our proposed Character-level word embedding allows us to obtain representations of even the rare words (without storing the embedding for these words), and also captures local subword information for each word.

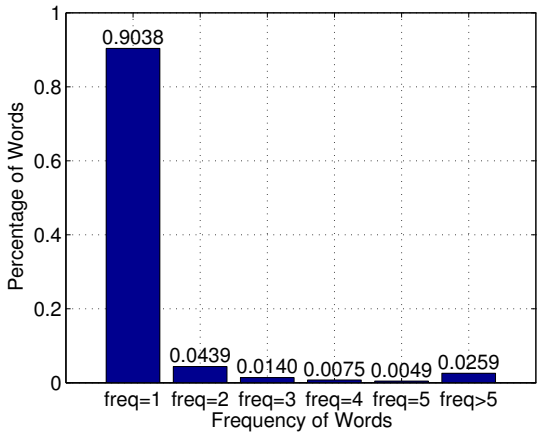

Figure 3: Word Frequency Distribution. More than 90% of the words appear only once in the entire training corpus. Storing all the words, and learning their embeddings during training can be computationally prohibitive.

4.2 Evaluation of URLNet

4.2.1 Experimental Settings and Baselines. We constructed 2 training corpuses: first a set of 1 million URLs randomly sampled from the 5 million, and the second includes the entire 5 million URLs. The sampling was done such that the proportion of malicious and benign URLs was still maintained.

As a baseline model, we used L1-regularized L2-loss SVM (implemented in Liblinear [11]). The SVM was trained on the 5 sets of baseline lexical features described in the previous subsection, according to [2, 25, 27]. The baseline methods include SVMs trained on: i) **Whole URL BoW**; (ii) **UCT** [27]; (iii) **PSB**[2]; (iv) **Character Trigrams** [25]; and (v) **Combined**: a combination of all these features. We compared the baselines with the proposed URLNet:

- **URLNet(Character-Level)** - only the Character-level CNN,
- **URLNet(Simple Word-Level)** - only the Word-level CNN,
- **URLNet(Full)** - which is the end-to-end framework combining both character-level and word-level CNNs (including special characters and character-level word embedding).

For Word-level CNN, the words obtained were based on the Whole URL BoW (i.e. there was no separate dictionaries for different parts of the URL). All Deep Learning models were implemented using both Keras [8] and Tensorflow [1][2], and optimized using the Adam Optimizer[19]. A Dropout rate of 0.5 was used in the convolutional layers. All the models were tested on the Test corpus of 10 million URLs and were evaluated on the basis of Area under the ROC Curve (AUC) (due to the imbalanced nature of the dataset). In addition, we also compare model performance in terms of True Positive Rates at different levels of False Positive Rate to observe the detection rate of malicious URLs at a given false positive rate (or false alarm rate).

4.2.2 Results. The main results of this paper can be seen in Table 3, and we can further visualize the AUC characteristics in Figure 4. In general URLNet based methods significantly outperform the baseline methods across all metrics (AUC and TPR@FPR). Within the baseline models, we observe that heuristically accounting for sequential information through separate dictionaries in URL Component Tokenization (UCT) is able to improve ther performance over using simple Whole URL BoW Features. Similarly, using other advanced and expert features (PSB and Character Trigrams) is able to give incremental improvement, and the best baseline model is obtained by combining all the features together.

In contrast, without using any expert or hand-designed features, URLNet methods offer a significant jump in AUC over baselines. It

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

is clear that the proposed URLNet is able to capture several types of semantic and structural information in the URL, which existing methods based on bag-of-words features could not. Within URLNet, we observe the performance of the 3 variants: Character-Level, word-level, and Full. While Character-level and Word-Level URLNet have similar performances, URLNet(Full) largely exploits the positives of both, and provides a more consistently better performance. At low FPRs, word-level URLNet has a better performance than Character-level URLNet, while at higher FPRs, the reverse holds. URLNet(Full) combines the merits of both, and except at FPR=0.0001, it gives a better performance in all other scenarios, including offering a significant boost to the AUC. On the whole, we also observe that increasing training data size from 1 million to 5 million has a positive impact across all metrics.

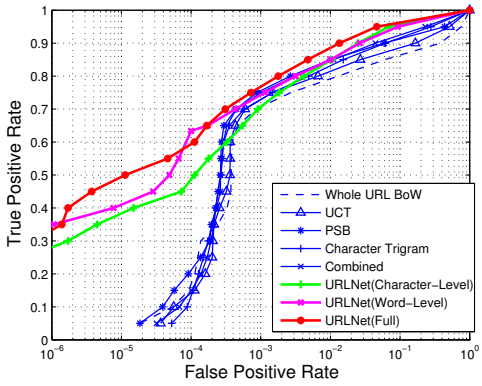

Figure 4: Area Under ROC Curve (Trained on 1m, Tested on 10m). URLNet(Full) is slightly worse than URLNet(Word-level) at FPR = 10^{-4}, but better otherwise. URLNet(Full) is consistently better than URLNet(Character-level). All URLNet variants outperform baselines.

4.3 Ablation Analysis

The performance of different components of URLNet, and the incremental performance gain of each component can be seen in Table 4. Within URLNet (Word-Level), treating special characters as words and using character-level word embedding improve the AUC score. This validates the usage of special characters as words and the benefits of character-level word embedding. Even though this improvement is consistent with varying training data size, a minor discrepancy occurs at the low levels of FPR, where Simple-Word Level URLNet offers the highest TPR. A possible reason for this is that using special characters as words, and using a character-level word embedding make the performance of the word-level CNN slightly resemble that of Character-level CNN, which is why the performance at high FPRs improves, while at low FPRs, it worsens a bit. The trend of word-level URLNet is carried over to URLNet(Full) where we see steady improvement in AUC as we integrate additional features one at a time (special characters as words, and character-level words). On the whole, URLNet(Full) exploits character-level and word-level information and significantly outperforms both Character-based URLNet and Word-based URLNet.

4.4 Visualization

In this section, we visualize the embedding features of URLs extracted from the proposed URLNet model and compare with one of the baselines. We randomly sampled 2,000 URLs from the test dataset with balanced distribution of classes (1,000 Benign and 1,000 Malicious) and extracted the feature vectors of these URLs from one of the layers in URLNet (trained on 5 million URLs). Here we selected the feature vectors after concatenation of the outputs of char-level and word-level CNN branches (*Concatenated Char and Word feature vector* layer in Figure 2), and obtained a 1,024-dimensional vector. For the baseline features, we extract BoW features and expert features being used in the *Character Trigrams*[25]. The baseline feacture vector is 14,604-dimensional. From the extracted feature vectors, we apply t-SNE [28] to reduce feature dimension and plot the URLs on a 2-dimensional embedding space. The Figure of the embedded URLs can be seen in Figure 5 and Figure 6.

As can be seen in Figure 5, for URLNet, the malicious URLs and benign URLs are clearly seperated into two groups of URLs. Most of Benign URLs are located in the left area of the plot while Malicious URLs are in the right area. Very few data points of Malicious URLs are overlapping with Benign URLs. In the baseline embedding space (Figure 6), the seperation between malicious and benign URLs is not as clear. Many Benign and Malicious URLs are located in the center of the plot and are overlapping with each other. Different from URLNet, in the baseline embedding, many individual data points are scattered around the plot, making it hard to identify which clusters those data points are more likely to belong to.

In addition to Benign/Malicious separation, we also observe several clusters appearing in the plots. We further analyze some of these clusters to identify potential patterns in the URL strings which may possibly be indicative of malicious or benign nature of a URL. We highlight some of the data points with different markers (in black), each of which indicates a type of lexical pattern in the URL string. Refer to Table 5 for the details of the lexical patterns and example URLs. Analysis of such patterns could be useful in a deeper understanding of the Malicious URL properties. For example, URLs that contain phrases such as 'tumblr' in URL domain, '/opt/' in the URL path, or '.exe' file extension, are clustered together. For phrases that appear at different parts of the URL such as 'google', we are still able to distinguish two clusters of URLs, one for 'google' in the URL domain, and one for 'google' in the URL path, as can be seen in Figure 5, despite not making any distinction between different URL components during training. Similarly, without usage of expert features, URLNet obtains meaningful representation and embeds those URLs where primary domain contains an IP together; or if the length of PD > 10, the URLs get clustered together.

Finally, with smaller number of dimensions than the baseline feature vector and without the need to obtain expert features, URLNet feature vector is lightweight and efficient to represent the URL. For data size of 2,000 URLs, URLNet feature vector is about 14 times smaller than the baseline feature vector. Hence, URLNet does not suffer from the memory constraints to process and store the vectors which can be used for further downstream tasks. In contrast to the baselines, the number of dimensions of URLNet feature vector does not vary by the data size, making it easy to obtain and process embedding features of large scale datasets (millions of URLs).

Table 3: Evaluation of URLNet compared to baselines and special cases of URLNet: Character-level CNN and Word-Level CNNs. All results are on the Test Data of 10 million URLs. The best performances have been bolded.

	Training Size = 1m					Training Size = 5m				
	TPR @ FPR Level					TPR @ FPR Level				
	0.0001	**0.001**	**0.01**	**0.1**	**AUC**	**0.0001**	**0.001**	**0.01**	**0.1**	**AUC**
Baselines										
Whole URL BoW	0.1759	0.7124	0.7915	0.8711	0.9208	0.3189	0.7714	0.8436	0.8995	0.9387
UCT [27]	0.1431	0.7292	0.8163	0.8890	0.9400	0.2856	0.7890	0.8624	0.9090	0.9548
PSB [2]	0.2151	0.7560	0.8500	0.9126	0.9549	0.1750	0.8179	0.8834	0.9291	0.9656
Character Trigrams[25]	0.1347	0.7326	0.8322	0.9161	0.9611	0.2017	0.7929	0.8682	0.9275	0.9665
Combined	0.1490	0.7499	0.8457	0.9259	0.9669	0.2039	0.8218	0.8883	0.9423	0.9736
URLNet										
URLNet (Character-level)	0.5159	0.6876	0.8370	0.9546	0.9828	0.6463	0.7824	0.8991	0.9735	0.9892
URLNet (Word-level)	**0.6337**	0.7524	0.8443	0.9297	0.9730	**0.7312**	0.8168	0.8878	0.9595	0.9842
URLNet (Full)	0.5849	**0.7683**	**0.8860**	**0.9722**	**0.9885**	0.7160	**0.8248**	**0.9084**	**0.9858**	**0.9929**

Table 4: Ablation Analysis: We compare the performance gain from each critical component proposed in URLNet. Both URLNet(Character-level) and URLNet(Word-level) are competitive, and are outperformed by URLNet(Full).

	Training Size = 1m					Training Size = 5m				
	TPR @ FPR Level					TPR @ FPR Level				
	0.0001	**0.001**	**0.01**	**0.1**	**AUC**	**0.0001**	**0.001**	**0.01**	**0.1**	**AUC**
URLNet (Character-level)										
↪ Character-level CNN	0.5159	0.6876	0.8370	0.9546	0.9828	0.6463	0.7824	0.8991	0.9735	0.9892
URLNet (Word-level)										
Word CNN	**0.6337**	0.7524	0.8443	0.9297	0.9730	**0.7312**	**0.8168**	0.8878	0.9595	0.9842
↪ + Special Characters as Words	0.5957	0.7425	0.8496	0.9537	0.9832	0.7179	0.8108	0.8832	0.9649	0.9853
↪ + Character-Level Words	0.6172	**0.7694**	**0.8792**	**0.9664**	**0.9865**	0.6937	0.8007	**0.8973**	**0.9673**	**0.9878**
URLNet (Full)										
↪ Character + Word	**0.6239**	0.7565	0.8722	0.9620	0.9864	**0.7261**	0.8169	0.8999	0.9769	0.9907
↪ + Special Character Words	0.6008	0.7512	0.8634	0.9643	0.9853	0.6901	0.8104	0.8996	0.9797	0.9918
↪ + Character-level Words	0.5849	**0.7683**	**0.8860**	**0.9722**	**0.9885**	0.7160	**0.8248**	**0.9084**	**0.9858**	**0.9929**

Table 5: Examples of lexical patterns in URLs and example URLs. The lexical patterns are extracted at different parts of the URL string: primary domain, URL path, and file extension.

URL Component	**Lexical Pattern**	**Example URL**
Primary Domain	contains 'tumblr'	http://exampledomain.tumblr.com/
	contains 'google'	http://www.google.com/urlpath/... http://abcd123googlexyz456.com/urlpath/...
	contains IP	http://192.168.0.1/ http://192.168.0.1/urlpath/...
	has average word length >10	http://a1ds2dce0b33fdgd425d8fsgg9836c4234d0.exampledomain.net/
Path	contains 'google'	http://www.exampledomain.com/filename?f=GOOGLEEARTH... http://exampledomain.net/urlpath/googledrive/sub_dir/...
	contains '/opt/'	http://www.exampledomain.com/opt/...
	contains the *dash* pattern in the last path token	http://exampledomain.com/urlpath/abc-123-fff-456-...
File Extension	Includes a file with extension 'exe'	http://exampledomain.net/urlpath/filename.exe
	Includes a file with extension 'zip'	http://exampledomain.com/urlpath/filename.zip

5 RELATED WORK

For a comprehensive review on Malicious URL Detection using Machine Learning, see [33]. Here, we briefly discuss two most related topics for our work: Feature Representation for Malicious URL Detection, and Deep Learning (particularly for Natural Language processing).

5.1 Feature Representation for Malicious URL Detection

Before training a prediction model, a raw URL is typically converted to a suitable feature vector $\mathbf{u} \rightarrow \mathbf{x}$, so that it can be interpreted by traditional machine learning models. This feature representation needs to be chosen carefully, as the classification models work on the premise that distributions of the features for malicious and benign URLs are different. Researchers have proposed several types of features for this task, including blacklist features [12, 26], lexical features [20, 26, 27, 41], host-based features [5, 26, 30], content features [3, 37, 40], and context and popularity based features [4, 7, 24]. Blacklist features use the presence of a URL in a blacklist as a feature, as they could be strong indicators. Lexical Features focus on string properties of the URL, e.g. length of URL, number of special characters, types of words that appear in the URL string, alphanumeric distribution of characters, etc. Host-based features are those derived from the host-name properties of the URL including information such as IP Address, WHOIS information, Geographic location, etc. Content features are those that require explicitly visiting and downloading the content hosted by the URL, in order to obtain information such as HTML and JavaScript features. Context and popularity features correspond to information about where the URLs have been shared on social media, or their ranking and popularity scores. Many researchers have used a combination of some of these features, which was often determined through expert domain knowledge.

Obtaining features from URLs can be an expensive task from the perspective of security threats and engineering overhead. For example, obtaining content-based features can be very slow, and at the same time is highly risky. Moreover, identifying which features are useful requires expert domain knowledge. Consequently, usage of information directly obtainable from the raw URL was popularized [26, 27]. It was shown that lexical features, which are the easiest to obtain, gave competitive performances [2, 26]. In our work we focused primarily on the lexical features to obtain the feature representation for the URLs. Among lexical features, various types of information can be obtained from the URL. While some statistical properties of the URL string, such as length of the URL, number of dots, etc. [20] have been used, the most popular features were Bag of Words, Term Frequency features or n-gram features [2, 20, 26]. However, none of these methods effectively capture the sequential properties of the URL string (or substrings). Moreover these methods fail to extract useful information from unseen words in the test URLs.

There have been other advanced lexical features used, such as Kolmogorv Complexity [32], obfuscation resistant features [22], intra-url relatedness [29], etc. However, they required substantial feature engineering or expert knowledge, or they were not scalable to millions of URLs, thus reducing their practical applicability.

5.2 Deep Learning

Deep Learning or Representation Learning has received increasing interest in recent years owing to their success in several applications [13, 14, 21, 23, 35]. The core idea is to automatically learn the feature representation from raw or unstructured data, in an end-to-end manner without using any hand designed features. Following this principle, we aim to use Deep Learning for Malicious URL Detection, in order to directly learn representation of the raw URL string, without using any hand designed expert features.

Since we aim to train Deep Networks over lexical features, a closely related area is Deep Learning for Natural Language Processing (NLP). Deep learning methods have found success in many NLP tasks: text classification [18], machine translation [6], question answering [38], etc. Recurrent neural networks (e.g. LSTM [15]) have been widely used due to their ability in capturing sequential information. However, the problems of exploding and vanishing gradients is magnified for them, making them difficult to train. Recently, Convolutional Neural Networks have become excellent alternatives to LSTMs, in particular showing promising performance for text classification using Word-level CNNs [18] and Character-level CNNs [39].

There have been very limited attempts at using Deep Learning for Malicious URL Detection. We recently noticed a work parallel to ours [34] that attempted to use Character-level CNNs for this task. However, they ignored several types of structural information that could be captured by words in the URLs. In contrast to their work, (i) we consider both word-level and character-level information; (ii) through extensive analysis we show the importance of word-level information in capturing longer temporal patterns; (iii) we develop novel character-level word embedding for effectively utilizing word-level information - in particular handling the presence of too many unique words, and obtaining embeddings for unseen words at test time; and (iv) we train the whole model in a jointly optimized framework. URLNet comprehensively captures the structural information available in the URL String through both character and word-level information. In fact, [34] is a special case of our proposed URLNet, where only character-level information is considered. Further, we have even shown that URLNet(Full) is consistently better than just a Character-level URLNet in AUC (and TPR at all levels of FPR - particularly at low FPRs).

6 CONCLUSION

In this paper we proposed URLNet, a CNN based deep neural network for Malicious URL Detection. Existing approaches mostly used Bag of Words like features, and this caused them to suffer from some critical limitations, including inability to detect sequential concepts in a URL string, requiring manual feature engineering, and inability to handle unseen features in test URLs. We proposed Character CNNs and Word CNNs for this task, and jointly optimized the network. Moreover, we proposed advanced word-embedding techniques which are particularly useful to deal with rare words, a problem usually observed in malicious URL Detection tasks (and not in traditional NLP tasks). This approach also allowed URLNet to learn embeddings from unseen words at test time, and exploit subword information. Our approach worked in an end to end manner without requiring any expert features.

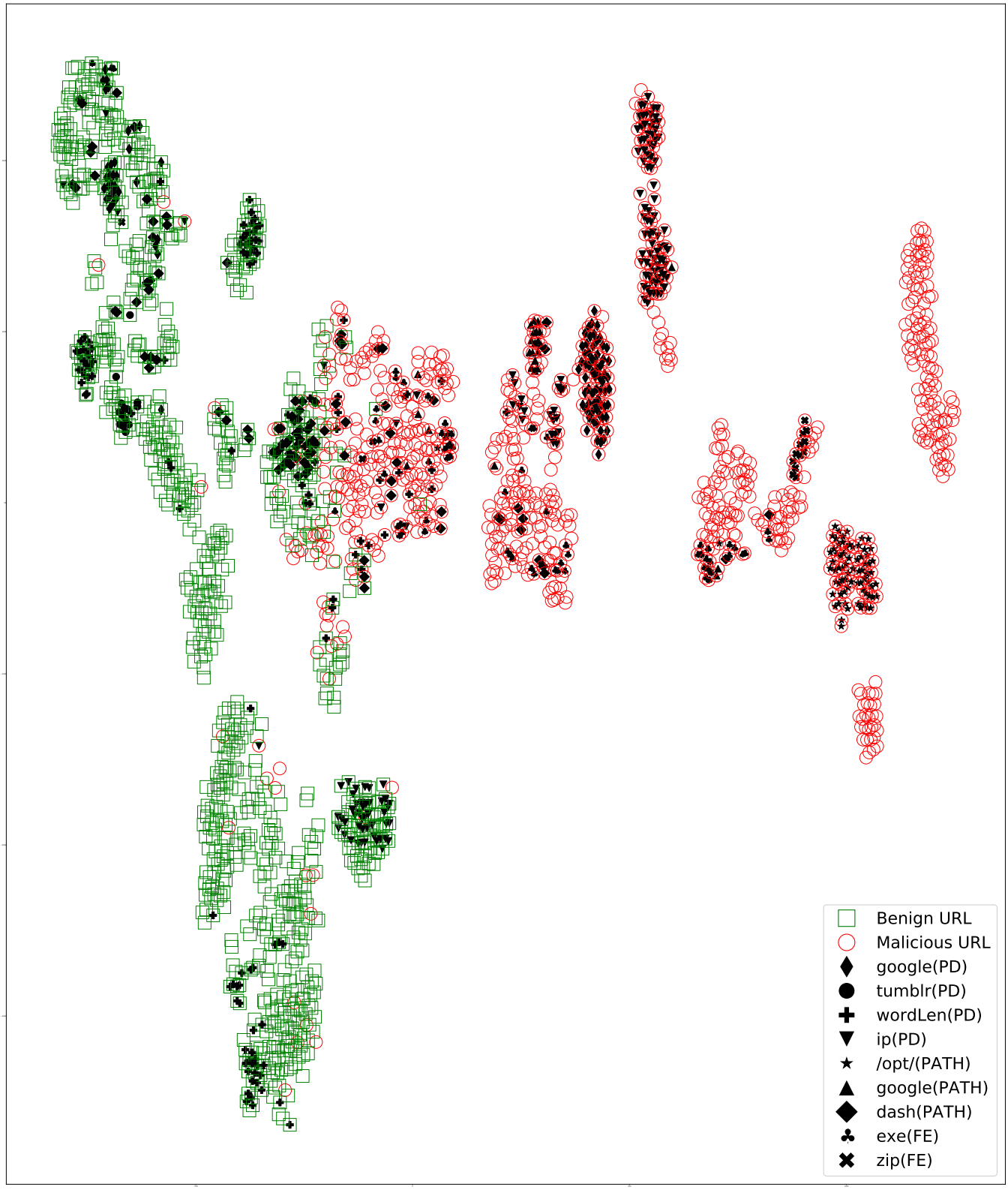

Figure 5: Visualization of feature embedding of sampled URLs using URLNet features. The data points are color-coded by the URL classes: Benign (Green) and Malicious (Red). The markers are also configured to distinguish certain types of lexical patterns in URL components: PD (Primary Domain), PATH (URL Path), and FE (File Extension).

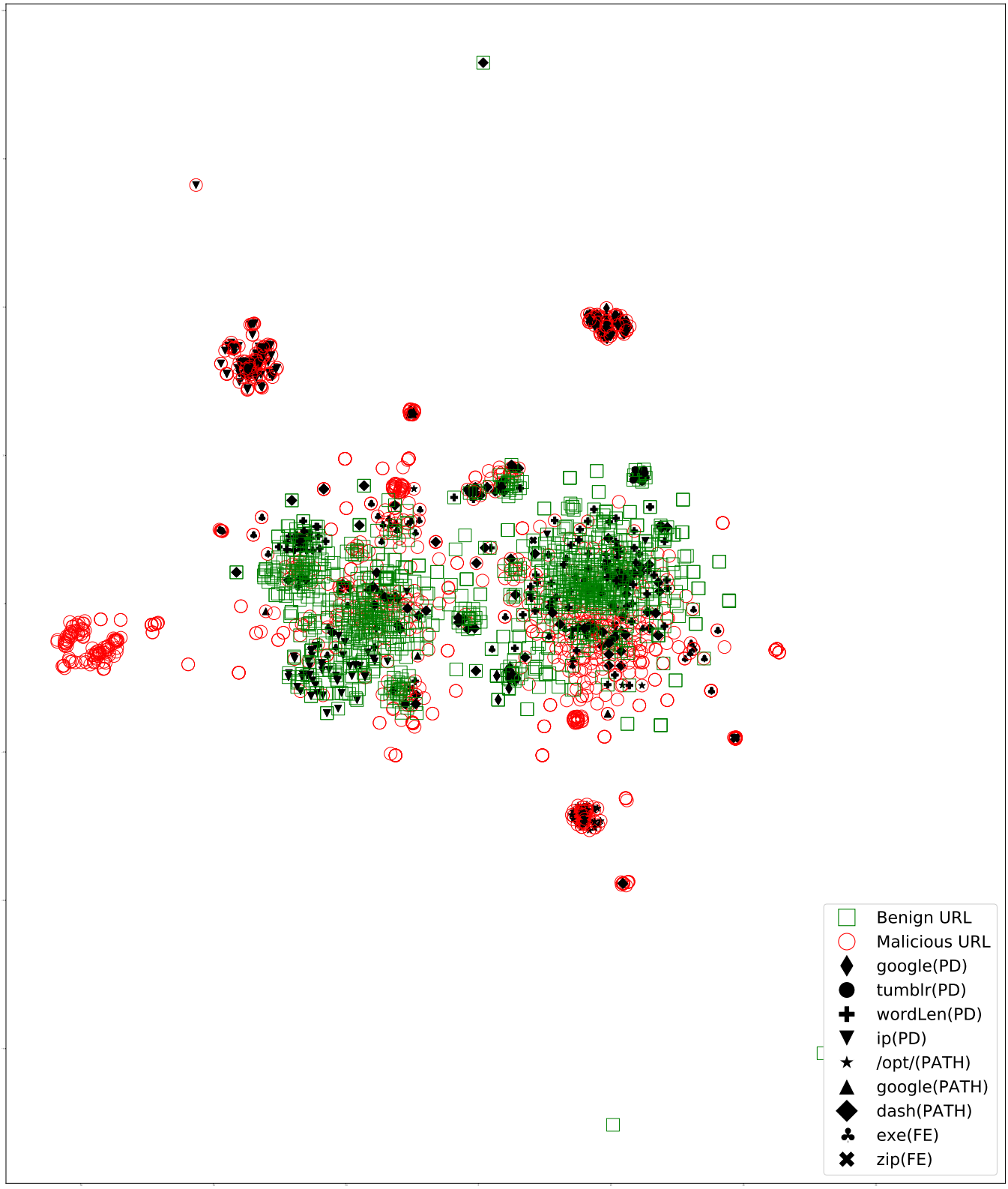

Figure 6: Visualization of feature embedding of sampled URLs using Character Trigrams[25] features. The data points are color-coded by the URL classes: Benign (Green) and Malicious (Red). The markers are also configured to distinguish certain types of lexical patterns in URL components: PD (Primary Domain), PATH (URL Path), and FE (File Extension).

ACKNOWLEDGEMENTS

We are very grateful to our collaborators VirusTotal, for providing us access to their data, without which this research would not have been possible.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[2] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. 2010. Lexical feature based phishing URL detection using online learning. In *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*. ACM, 54–60.

[3] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. 2011. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th international conference on World wide web*. ACM, 197–206.

[4] Jian Cao, Qiang Li, Yuede Ji, Yukun He, and Dong Guo. 2014. Detection of Forwarding-Based Malicious URLs in Online Social Networks. *International Journal of Parallel Programming* (2014), 1–18.

[5] Daiki Chiba, Kazuhiro Tobe, Tatsuya Mori, and Shigeki Goto. 2012. Detecting malicious websites by learning ip address features. In *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on*. IEEE, 29–39.

[6] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[7] Hyunsang Choi, Bin B Zhu, and Heejo Lee. 2011. Detecting malicious web links and identifying their attack types. In *Proceedings of the 2nd USENIX conference on Web application development*. USENIX Association, 11–11.

[8] François Chollet et al. 2015. Keras. (2015).

[9] Weibo Chu, Bin B Zhu, Feng Xue, Xiaohong Guan, and Zhongmin Cai. 2013. Protect sensitive sites from phishing attacks using features extractable from inaccessible phishing URLs. In *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 1990–1994.

[10] Cícero Nogueira Dos Santos and Maira Gatti. 2014. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts.. In *COLING*. 69–78.

[11] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research* 9 (2008), 1871–1874.

[12] Mark Felegyhazi, Christian Kreibich, and Vern Paxson. 2010. On the Potential of Proactive Domain Blacklisting. *LEET* 10 (2010), 6–6.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385* (2015).

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[16] Jason Hong. 2012. The state of phishing attacks. *Commun. ACM* 55, 1 (2012), 74–81.

[17] Rie Johnson and Tong Zhang. 2014. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058* (2014).

[18] Yoon Kim. 2014. Convolutional neural networks for sentence classification. *EMNLP* (2014).

[19] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[20] Pranam Kolari, Tim Finin, and Anupam Joshi. 2006. SVMs for the Blogosphere: Blog Identification and Splog Detection. In *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*. 92–99.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[22] Anh Le, Athina Markopoulou, and Michalis Faloutsos. 2011. Phishdef: Url names say it all. In *INFOCOM, 2011 Proceedings IEEE*. IEEE, 191–195.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[24] Sangho Lee and Jong Kim. 2012. WarningBird: Detecting Suspicious URLs in Twitter Stream.. In *NDSS*.

[25] Min-Sheng Lin, Chien-Yi Chiu, Yuh-Jye Lee, and Hsing-Kuo Pao. 2013. Malicious URL filteringâĂŤA big data application. In *Big Data, 2013 IEEE International Conference on*. IEEE, 589–596.

[26] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1245–1254.

[27] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 681–688.

[28] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[29] Samuel Marchal, Jérôme François, Radu State, and Thomas Engel. 2014. PhishScore: Hacking phishers' minds. In *Network and Service Management (CNSM), 2014 10th International Conference on*. IEEE, 46–54.

[30] D Kevin McGrath and Minaxi Gupta. 2008. Behind Phishing: An Examination of Phisher Modi Operandi. *LEET* 8 (2008), 4.

[31] LLC OpenDNS. 2016. PhishTank: An anti-phishing site. *Online: https://www.phishtank. com* (2016).

[32] Hsing-Kuo Pao, Yan-Lin Chou, and Yuh-Jye Lee. 2012. Malicious URL detection based on Kolmogorov complexity estimation. In *Proceedings of the The 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*. IEEE Computer Society, 380–387.

[33] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. 2017. Malicious URL Detection using Machine Learning: A Survey. *arXiv preprint arXiv:1701.07179* (2017).

[34] Joshua Saxe and Konstantin Berlin. 2017. eXpose: A Character-Level Convolutional Neural Network with Embeddings For Detecting Malicious URLs, File Paths and Registry Keys. *arXiv preprint arXiv:1702.08568* (2017).

[35] Jürgen Schmidhuber. 2015. Deep learning in neural networks: An overview. *Neural networks* 61 (2015), 85–117.

[36] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[37] Guang Xiang, Jason Hong, Carolyn P Rose, and Lorrie Cranor. 2011. Cantina+: A feature-rich machine learning framework for detecting phishing web sites. *ACM Transactions on Information and System Security (TISSEC)* 14, 2 (2011), 21.

[38] Caiming Xiong, Stephen Merity, and Richard Socher. 2016. Dynamic memory networks for visual and textual question answering. In *International Conference on Machine Learning*. 2397–2406.

[39] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*. 649–657.

[40] Yue Zhang, Jason I Hong, and Lorrie F Cranor. 2007. Cantina: a content-based approach to detecting phishing web sites. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 639–648.

[41] Peilin Zhao and Steven CH Hoi. 2013. Cost-sensitive online active learning with application to malicious URL detection. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 919–927.