Use of Supervised Machine Learning to Detect Abuse of COVID-19 Related Domain Names

Zheng Wang*

National Institute of Standards and Technology, 100 Bureau Drive, Gaithersburg, 20899, MD, USA

ARTICLE INFO

Keywords:
COVID-19
malicious domain name
classification
machine learning
deep learning

ABSTRACT

A comprehensive evaluation of supervised machine learning models for COVID-19 related domain name detection is presented. One representative conventional machine learning implementation and nineteen state-of-the-art deep learning implementations are evaluated. The deep learning implementation architectures evaluated include the recurrent, convolutional, and hybrid models. The detection rate metrics and the computing time metrics are considered in the evaluation. The result reveals that advanced deep learning models outperform conventional machine learning models in terms of detection rate. The results also show evidence of a tradeoff between detection rate and computing speed for the selection of machine learning models/architectures. High-frequency lexical analysis is provided for a better understanding of the COVID-19 related domain names. The limitations, implications, and considerations of the use of supervised machine learning to detect abuse of COVID-19 related domain names are discussed.

1. Introduction

With the worldwide dissemination of COVID-19, individuals have been trying to acquire vital data and directives from specific websites. Coronavirus-related domain name registrations have been growing explosively, accompanying the rise in user interest. A study found a 656 percent rise in the average daily domain name registrations associated with coronavirus from February to March in 2020 [1]. At the same time, the public demand for coronavirus-related knowledge, goods, and services has been exploited by some unscrupulous cybercriminals for cyberattacks. Researchers also found a 569% rise in malicious registrations and a 788% growth in high-risk registrations [1] during the period from February to March in 2020. The COVID-19 related violations found by malicious or high-risk domain names include phishing, malware, scams, illegal coin mining, domains with proof of malicious URL (Uniform Resource Locator) association, etc.

Identifying COVID-19 related domain names is typically challenging because of the limited access to some resources and the high cost of implementation and deployment. A variety of traits exploited in the existing malicious domain name detection approaches are based on unique behavior reflected in different phases of the lifecycle of malicious domain names. Their data sources include DNS (Domain Name System) traffic data passively or actively collected from DNS clients, resolvers, or servers, geo-location information/database, ASN (Autonomous System Number), registration records, IP (Internet Protocol)/domain blocklists/allowlists, network data, etc. However, most of the data sources are not available for typical domain name users, registrants, and common small registrars. Also, the real-time implementation of many detection algorithms requires a non-trivial cost of IT resources which is not affordable for most stake-holders. That necessitates developing COVID-19 related domain name detection algorithms which are purely based on domain name strings and are lightweight and fast enough for real-time usage.

In this work, we consider the task of detecting COVID-19 related domain names as a binary classification task: given a domain name string as input, classify it as either COVID-19 related or benign. A machine learning approach, which depends completely on human crafted features, was proposed to address the malicious domain name detection problem [2]. Recent years also witnessed deep neural networks in the literature on malicious domain name detection [2, 3, 4, 5, 6, 7, 8, 9]. While deep neural networks were demonstrated to outperform traditional machine learning methods in terms of accuracy in some domains/tasks, it is not clear whether it is the same case for this specific task of combating abuse of COVID-19 related domain names. Also, there is not a systematic study providing a comparative

*This work was completed while a Research Associate at the National Institute of Standards and Technology.
zhengwang98@gmail.com (Z. Wang)
ORCID(s):

evaluation of a variety of deep learning architectures, leaving one to wonder which one has the best detection accuracy for COVID-19 related domain names. Last but not least, we consider the training speed and the prediction speed among the key metrics of our evaluation because they are critical to real-time applications. One may wonder what tradeoff between accuracy and speed may be considered if they run counter to each other.

To answer these open questions, we evaluate the performance of one representative machine learning algorithm and nineteen different deep learning architectures for the classification problem (see Table 3) of detecting COVID-19 related domain names. Our findings confirm that the deep learning architectures work better than the conventional machine learning algorithm in terms of detection rate. But the differences among the deep learning architectures are not as obvious as their advantages over the conventional machine learning algorithms. If taking the speed metrics into account, the deep learning architecture with the best detection rate is not the best in terms of computing speed. So some tradeoff between accuracy and speed is indeed necessary for the classifier design.

The rest of the paper is structured as follows. Section 2 presents an overview of supervised machine learning models for malicious domain name detection. Section 3 details the implementations of one conventional machine learning model and nineteen deep learning models. Our evaluation is presented in Section 4. Some discussion is given in Section 5. The paper concludes in Section 6.

2. Supervised machine learning models

In this section, we briefly describe the supervised machine learning models for malicious domain name detection which underpin the implementations presented in Section 3.

2.1. Conventional machine learning models

A machine learning model maps a collection of input features to the target. The purpose of the model is to capture a pattern between the input features and the target so that the model can accurately predict the target given new input features, where the target is unknown.

For any given data set, the goal is generally to develop a model capable of predicting with the highest degree of accuracy possible. There are several levers in machine learning that may greatly impact the model's performance, including the choice of algorithm, the hyperparameters configuration, the amount and quality of the data set, and the input features used to train the model.

In many situations, finding the optimal features that best fit into the mapping between the input and the target is important and challenging for the success of machine learning. For example, some given set of features in the collection of raw features for a dataset, more often than not, cannot provide enough, or the most relevant, information to train a performant model. So feature selection techniques are often developed to choose a set of the most informative features to gain some improvement in the prediction accuracy, the computing cost, or the model interpretability. In other instances, the performance of the model can be enhanced by feature engineering that translates one or more features into a different representation to provide the model with better information.

2.2. Deep learning models

Conventional machine learning models were largely dominated by the feature engineering approach until deep learning techniques started demonstrating better performance in various domains. Unlike conventional machine learning models, deep learning models incorporate the burden of feature design to the underlying learning system. From this perspective, a deep learning model is more of a fully trainable model beginning from raw input, e.g., domain name strings in our task, to the final output, e.g., the recognized classification in our task. Deep learning techniques can be classified into recurrent models and convolutional models as follows.

2.2.1. Recurrent models

Recurrent neural networks (RNNs) [10] are basically designed to encode information about the sequence directly within the architecture of the neural networks. Besides the vanilla architecture, there are also a number of RNN variants, such as long short-term memory (LSTM) [11] and gated recurrent unit (GRU) [12], proposed to address the challenges of training RNNs, e.g., the vanishing and the exploding gradient problems. Attention mechanisms further enhance recurrent models by capturing long-term dependencies to arbitrary lengths.

RNN. RNNs have a fixed number of parameters while they can process inputs with a variable length. In a RNN, the specific positions in the input sequence are mapped to their corresponding neural units which process the sequence

serially. The positions in the input sequence are ofen known as its time-stamps. In general, each time-stamp has an input vector at time t (e.g., the embedding encoded vector of the t th character) $\bar{x}_t$, a hidden state at time t $\bar{h}_t$, and an output vector at time t $\bar{y}_t$. Then, the hidden state at time t is given by a function of the input vector at time t and the hidden vector at time $(t-1)$[1]:

$$\begin{aligned} \bar{h}_t &= \tanh\left(W_{xh}\bar{x}_t + W_{hh}\bar{h}_{t-1}\right) \\ \bar{y}_t &= W_{hy}\bar{h}_t \end{aligned} \tag{1}$$

where W_{xh} is the input-hidden matrix, W_{hh} is the hidden-hidden matrix, and W_{hy} is the hidden-output matrix.

LSTM. LSTM is an enhancement of the vanilla RNN architecture in which fine-grained control over the data written into this long-term memory is designed. The recurrence conditions of how the hidden states are propagated are changed by introducing an additional hidden vector namely the cell state. The recurrent equations are as follows:

$$\begin{bmatrix} \bar{i} \\ \bar{f} \\ \bar{o} \\ \bar{c} \end{bmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \quad W \begin{bmatrix} \bar{x}_t \\ \bar{h}_{t-1} \end{bmatrix} \tag{2}$$

$$\bar{c}_t = \bar{f} \odot \bar{c}_{t-1} + \bar{i} \odot \bar{c} \tag{3}$$

$$\bar{h}_t = \bar{o} \odot \tanh\left(\bar{c}_t\right) \tag{4}$$

where the update matrix is denoted by W which is used to multiply the column vector $\left[\bar{x}_t, \bar{h}_{t-1}\right]^T$ before the update. The four intermediate vector variables $\bar{i}$, $\bar{f}$, $\bar{o}$, and $\bar{c}$, which are known as input, forget, and output variables respectively, are used to update the states. The calculation of the hidden state vector $\bar{h}_t$ and the cell state vector $\bar{c}_t$ follows a multi-step process: first the four intermediate variables are computed and then the hidden variables are computed using the intermediate variables.

GRU. GRU can be viewed as a simplification of the LSTM, which does not use explicit cell states. Another difference is that the LSTM directly controls the amount of information changed in the hidden state using separate forget and output gates. On the other hand, a GRU uses a single reset gate to achieve the same goal. Thus GRU is simpler and enjoys the advantage of greater ease of implementation and efficiency.

In GRU, a vector W is multiplied to get a new vector that is transferred through the sigmoid activation to create two intermediate variables, $\bar{z}t$ and $\bar{r}t$ respectively. These two intermediate variables $\bar{z}t$ and $\bar{r}t$ are called upgrade and reset gates respectively. The hidden state vector $\bar{h}_t$ is caculated using the two gates and the weight matrix V: first compute the two gates, then update the gated hidden vector with the weight matrix V:

$$\begin{bmatrix} \bar{z} \\ \bar{r} \end{bmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \end{pmatrix} W \begin{bmatrix} \bar{x}_t \\ \bar{h}_{t-1} \end{bmatrix} \tag{5}$$

$$\bar{h}_t = \bar{z} \odot \bar{h}_{t-1} + (1 - \bar{z}) \odot \tanh V \begin{bmatrix} \bar{h}_t \\ \bar{r} \odot \bar{h}_{t-1} \end{bmatrix} \tag{6}$$

Bidirectional recurrent networks. One disadvantage of recurrent networks is that the state at a particular time unit only has knowledge about the past inputs up to a certain point in a sequence, but it has no knowledge about future states. In certain applications like language modeling, the results are vastly improved with knowledge about both past and future states.

[1] Biases will not be specifically used in this paper (for simplicity) because they can be integrated with bias neurons.

In the bidirectional recurrent networks, we have separate hidden states $\bar{h}_t^{(f)}$ and $\bar{h}_t^{(b)}$ for the forward and backward directions. The recurrence output can be written as follows:

$$\bar{y}_t = W_{hy}\left[\bar{h}_t^{(f)}, \bar{h}_t^{(b)}\right] \tag{7}$$

Attention mechanism. The principle of attention mechanism is primarily inspired by visual attention: many animals compute suitable responses by focusing on some parts of a whole visual object. Following the same principle, neural networks with the attention mechanism work by paying more attention to the most pertinent piece of information and less attention to the irrelevant piece of information. That selective neural network processing has found success in speech recognition, text translation, visual object detection [13], etc.

In a general setting, an attention mechanism is a method that takes n arguments $y_1, ..., y_n$, and a context c. It return a vector z which is supposed to be the summary of the y_i, focusing on information linked to the context c. More formally, it returns weighted arithmetic mean of the y_i, and the weights are chosen according to the relevance of each y_i given the context c. Namely, the ouptput can be written as:

$$z = \sum_{j=1}^{n} \alpha_j y_j \tag{8}$$

where α_j is a weight computed for each y_j. The weightings α_j are computed by:

$$e_j = score\left(c, y_j\right), \alpha_j = \frac{\exp\left(e_j\right)}{\sum_{k=1}^{n} \exp\left(e_k\right)} \tag{9}$$

where $score\left(\cdot\right)$ is a relevance scoring function.

2.2.2. Convolutional models

Convolutional models have shown excellent performance on image classification, object detection, speech recognition, natural language processing, etc. The main concept of convolutional models is to obtain local features from input (usually an image) at higher layers and combine them into more complex features at the lower layers [14].

CNN. In convolutional neural networks (CNNs), the states in each layer are organized in a spatial fashion. For traditional image processing tasks, each layer in the convolutional network is a 3-dimensional grid structure with height, width, and depth. The parameters are grouped into 3-dimensional structural units, known as filters or kernels. The convolution operation positions the filter at any possible location in the image. The function is extracted by applying filters with strides and padding if necessary. Pooling is typically used to minimize dimensionality.

CNNs work especially well on computer vision problems due to their ability to work convolutionally, i.e. by extracting features from local input patches. The same properties that make CNNs the best option for computer vision tasks also make them very relevant for some structured sequence data processing. In this work, we apply 1D CNNs to processing a specific sequence of characters, namely the domain name.

Capsule networks. Despite all their benefits, CNNs have some drawbacks one of which is the focus on object existence, not localization. While CNNs learn higher-level concepts in deeper layers and reduce computational complexity by pooling, they lose the pose and location information of the object. Especially, the relative spatial relationships between features are lost. Capsule networks [15] were proposed as a solution to the weakness.

Capsule networks let each capsule represent some extraction of the object by configuring a small neural network and encode the extracted information as a vector. The vector represents the probability of identifying some feature as its length and the state of the identified feature as its direction. The vector design allows invariant probability and variant direction when an object changes its position. A capsule selectively forwards its output vector to the capsules in the next layer based on relevancy following the so-called "dynamic routing".

Specifically, each capsule's state s_j is calculated as the weighted sum of the matrix multiplication of output/prediction vectors of the capsules from the lower layer with the coupling coefficient c_{ij} between s_j and the respective lower-level capsule s_i.

$$\mathbf{s}_j = \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}, \quad \hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij}\mathbf{u}_i \tag{10}$$

In particular, capsules in the first capsule layer of a capsule network calculate their activation based on the input from the previous convolution layer. In this case, no coupling coefficient c_{ij} exists. As the capsule's output vector indicates the probability of having detected a certain feature, capsule s_j's output vector v_j is squashed, so that long vectors sum up to 1 max and short vectors are close to zero.

$$\mathbf{v}_j = \frac{\left\|\mathbf{s}_j\right\|^2}{1+\left\|\mathbf{s}_j\right\|^2}\frac{\mathbf{s}_j}{\left\|\mathbf{s}_j\right\|} \tag{11}$$

The coupling coefficients c_{ij} defines the activation routing between a capsule and all potential parent capsules in the next layer and sum to 1. The softmax-like calculation ensure that the most likely parent capsule gets the most of capsule s_j's output.

$$c_{ij} = \frac{\exp\left(b_{ij}\right)}{\sum_k \exp\left(b_{ik}\right)} \tag{12}$$

By following the presented calculations, the routing preferences between capsules and the prediction of next layer activations, capsule networks claim to address the CNNs' limitations stated above, especially modeling stronger feature relationships.

3. Model implementation

We implemented the following models proposed in the literature using the architectures and hyperparameters specified in their respective papers: LSTM [9], Bi-LSTM [16], Par-CNN [17], CoreDNS-CNN [18], CapsNet [19], and CNN-LSTM [20]. For any hyperparameters unspecified in the literature for the models above, we chose the hyperparameters that produced the best accuracy. Besides the models above, we also implemented other models following the concepts stated in Section 2. Their hyperparameters were also tuned to obtain the ones with the best accuracies.

3.1. Conventional machine learning models

FANCI. FANCI [2] was developed to detect a specific type of malicious domain names using a conventional machine learning approach. Like other conventional machine learning workflows, feature engineering and feature extraction are key to the success of FANCI. FANCI uses 21 features to distinguish malicious domain names from benign ones. The 21 features include three categories: structural features, linguistic features, and statistical features. FANCI is purely based on the textual features of domain name strings. That is, all of the features are engineered on the raw domain name strings, without the help of any contextual information such as domain name registration and resolution data. Classifiers such as random forest (RF) classifiers and support vector machine (SVM) classifiers can be operated on the 21 features to obtain the predicted class. The implementation of workflow is as follows:

1. *Feature construction*: Extract the 21 features for the dataset.

2. *Hyperparameter optimization*: Do the grid search over the hyperparameter spaces for the RF and SVM algorithms and find out the hyperparameter set with the best detection accuracy.

3. *Performance evaluation*: Use the best algorithm with the optimal hyperparameter set to evaluate.

3.2. Deep learning models

The following deep learning models are described following the forward direction from the input to the output.

3.2.1. Recurrent models

RNN. The simple RNN implementation uses the embedding layer that encodes each character as a real-valued vector in a high dimensional space, where the similarity between characters in terms of meaning translates to closeness in the vector space. The embedding layer is also used as the first layer in all of the following deep learning model

implementations except for CoreDNS-CNN. The next layer is the vanilla RNN layer with 128 memory units. Then a dropout layer follows which probabilistically sets the inputs to zero (here the chance is 0.5) as the means of reducing overfitting. Finally, because this is a classification problem we use a dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes (benign and malicious) in the problem. The architecture of RNN is:

- *Embedding Layer* (embeddings dim=128).
- *RNN Layer* (unit=128, activation=tanh).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

Bi-RNN. The core difference between Bi-RNN and RNN is the bidirectional processing of RNN. In the bidirectional RNN layer, Bi-RNN sets the fraction of the units to drop for the linear transformation of the inputs and the fraction of the units to drop for the linear transformation of the recurrent state both to 0.2. Like the dropout layer, the two dropouts inside the bidirectional RNN layer help to reduce overfitting. The architecture of Bi-RNN is:

- *Embedding Layer* (embeddings dim=50).
- *Bidirectional RNN Layer* (unit=128, activation=tanh, dropout=0.2, recurrent dropout=0.2).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

LSTM. The simple LSTM is implemented by replacing the vanilla RNN layer in RNN with the LSTM layer. The architecture of LSTM is:

- *Embedding Layer* (embeddings dim=128).
- *LSTM Layer* (unit=128, activation=tanh).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

Bi-LSTM. The bidirectional LSTM is constructed by replacing the bidirectional RNN layer in Bi-RNN with the bidirectional LSTM layer. The architecture of Bi-LSTM is:

- *Embedding Layer* (embeddings dim=50).
- *Bidirectional LSTM Layer* (unit=128, activation=tanh, dropout=0.2, recurrent dropout=0.2).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

GRU. The simple GRU is implemented by replacing the vanilla RNN layer in RNN with the GRU layer. The architecture of GRU is:

- *Embedding Layer* (embeddings dim=128).
- *GRU Layer* (unit=128, activation=tanh).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

Bi-GRU. The bidirectional GRU is constructed by replacing the bidirectional RNN layer in Bi-RNN with the bidirectional GRU layer. The architecture of Bi-GRU is:

- *Embedding Layer* (embeddings dim=50).
- *Bidirectional GRU Layer* (unit=128, activation=tanh, dropout=0.2, recurrent dropout=0.2).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

Stack-LSTM. In the stacked LSTM, we implement an LSTM model comprised of two LSTM layers. The upstream LSTM layer provides a sequence output rather than a single value output to the downstream LSTM layer. Specifically, one output per input time step, rather than one output time step for all input time steps. The intuition is that stacked LSTM layers allow for greater model complexity and therefore can capture abstract concepts in the sequences, which might be helpful for some tasks. The architecture of Stack-LSTM is:

- *Embedding Layer* (embeddings dim=128).
- *LSTM Layer* (unit=128, activation=tanh).
- *LSTM Layer* (unit=128, activation=tanh).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

Stack-Bi-LSTM. We can obtain the stacked bidirectional LSTM by replacing the stacked LSTM layers in Stack-LSTM with the stacked bidirectional LSTM layers. The architecture of Stack-Bi-LSTM is:

- *Embedding Layer* (embeddings dim=50).
- *Bidirectional LSTM Layer* (unit=128, activation=tanh, dropout=0.2, recurrent dropout=0.2).
- *Bidirectional LSTM Layer* (unit=128, activation=tanh, dropout=0.2, recurrent dropout=0.2).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

Bi-LSTM-Att. In our implementation of the bidirectional LSTM with the attention mechanism, a context vector $\bar{c}_t$ is created as the weighted average of the bidirectional hidden vectors: where the attention variable $\alpha(t, j)$ is computed as:

$$e(t,j) = v_a^T \tanh\left(\left[\bar{h}_j^{(f)}, \bar{h}_j^{(b)}\right]\right), \alpha(t,j) = \frac{\exp\left(e_j\right)}{\sum_{k=1}^{n} \exp\left(e_k\right)} \tag{13}$$

Therefore, we create a new target hidden state $\bar{H}_t$ that combines the information in the context and the original bidirectional hidden state as follows:

$$\bar{H}_t = \tanh\left(W_c \begin{bmatrix} \bar{c}_t \\ \bar{h}_t^{(f)} \\ \bar{h}_t^{(b)} \end{bmatrix}\right) \tag{14}$$

The architecture of Bi-LSTM-Att is:

- *Embedding Layer* (embeddings dim=50).
- *Bidirectional LSTM Layer* (unit=128, activation=tanh).
- *Attention Layer*
- *Dropout Layer* (dropout=0.2).
- *Dense Layer* (unit=1, activation=sigmoid).

3.2.2. Convolutional models

CNN. The shallow CNN has only one CNN layer. The architecture of CNN is:

- *Embedding Layer* (embeddings dim=128).
- *Dropout Layer* (dropout=0.2).
- *1D CNN Layer* (filters=250, kernel size=3, strides=1, padding=no, activation=relu).
- *Flatten Layer*
- *Dense Layer* (units=250).
- *Dropout Layer* (dropout=0.2).
- *ReLU Layer*.
- *Dense Layer* (unit=1, activation=sigmoid).

CNN-Maxpool. The CNN with max pooling is implemented with two CNN layers. The upstream CNN layer is followed by a max pooling layer. We also add batch normalization layers to accelerate training and provide some regularization. The architecture of CNN-Maxpool is:

- *Embedding Layer* (embeddings dim=50).
- *Dropout Layer* (dropout=0.25).
- *1D CNN Layer* (filters=250, kernel size=4, strides=1, padding=same[2], activation=relu).
- *1D Max Pooling Layer* (pool size=3, padding=no).
- *1D CNN Layer* (filters=250, kernel size=3, strides=1, padding=same, activation=relu).
- *Flatten Layer*
- *Batch Normalization Layer*
- *Dense Layer* (units=250).
- *Dropout Layer* (dropout=0.2).
- *Batch Normalization Layer*
- *Dense Layer* (unit=1, activation=sigmoid).

CNN-Avepool. The CNN with average pooling is obtained by switching the max pooling layer in CNN-Maxpool to the average pooling layer. The architecture of CNN-Avepool is:

- *Embedding Layer* (embeddings dim=50).
- *Dropout Layer* (dropout=0.25).
- *1D CNN Layer* (filters=250, kernel size=4, strides=1, padding=same, activation=relu).
- *1D Average Pooling Layer* (pool size=3, padding=no).
- *1D CNN Layer* (filters=250, kernel size=3, strides=1, padding=same, activation=relu).
- *Flatten Layer*

[2]Padding evenly to the left/right of the input such that output has the same dimension as the input.

- *Batch Normalization Layer*
- *Dense Layer* (units=250).
- *Dropout Layer* (dropout=0.2).
- *Batch Normalization Layer*
- *Dense Layer* (unit=1, activation=sigmoid).

Stack-CNN-Maxpool. In the stacked CNNs with max pooling, we add one more CNN layer to CNN-Maxpool to obtain deeper CNNs. The architecture of Stack-CNN-Maxpool is:

- *Embedding Layer* (embeddings dim=50).
- *Dropout Layer* (dropout=0.25).
- *1D CNN Layer* (filters=250, kernel size=4, strides=1, padding=same, activation=relu).
- *1D Max Pooling Layer* (pool size=3, padding=no).
- *1D CNN Layer* (filters=300, kernel size=3, strides=1, padding=same, activation=relu).
- *1D Max Pooling Layer* (pool size=3, padding=no).
- *1D CNN Layer* (filters=350, kernel size=2, strides=1, padding=same, activation=relu).
- *Flatten Layer*
- *Batch Normalization Layer*
- *Dense Layer* (units=250).
- *Dropout Layer* (dropout=0.2).
- *Batch Normalization Layer*
- *Dense Layer* (unit=1, activation=sigmoid).

CoreDNS-CNN. CoreDNS implements a module of malicious domain name detection [18]. It borrows the idea of convolutional models for 2D image processing. To map the 1D input to the 2D space, it transforms a 256-character input to a 16×16 array. Thus a 2D input is processed like an image and the task is considered as a classic image classification problem. The architecture of CoreDNS-CNN is:

- *2D CNN Layer* (filters=16, kernel size=(2,2), strides=(1,1), padding=no, activation=relu).
- *2D Max Pooling Layer* (pool size=(2, 2), padding=no).
- *2D CNN Layer* (filters=16, kernel size=(2,2), strides=(1,1), padding=no, activation=relu).
- *2D Max Pooling Layer* (pool size=(2, 2), padding=no).
- *2D CNN Layer* (filters=8, kernel size=(2,2), strides=(1,1), padding=no, activation=relu).
- *Flatten Layer*
- *Dense Layer* (units=8, activation=relu).
- *Dense Layer* (unit=1, activation=sigmoid).

Par-CNN. The parallel CNNs are implemented by building multiple kernels, e.g., with different kernel sizes here, in one CNN layer resulting in multiple channel outputs per layer. Instead of deeper networks gained by stacking layers, this would result in wider networks based on the hope that the network learns a varied set of intermediate features. The architecture of Par-CNN is:

- *Embedding Layer* (embeddings dim=128).
- ***Concatenate*[***1D CNN Layer* (filters=256, kernel size=2, strides=1, padding=same, activation=relu), *1D CNN Layer* (filters=256, kernel size=3, strides=1, padding=same, activation=relu), *1D CNN Layer* (filters=256, kernel size=4, strides=1, padding=same, activation=relu), *1D CNN Layer* (filters=256, kernel size=5, strides=1, padding=same, activation=relu)**]**.
- *Dense Layer* (units=1024, activation=relu).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (units=1024, activation=relu).
- *Dropout Layer* (dropout=0.5)..
- *Dense Layer* (unit=1, activation=sigmoid).

CapsNet. Following the concept of capsule networks introduced in Section 2.2.2, the architecture of CapsNet is:

- *Embedding Layer* (embeddings dim=128).
- *1D CNN Layer* (filters=256, kernel size=8, strides=1, padding=no, activation=relu).
- *Dropout Layer* (dropout=0.5).
- *1D CNN Layer* (filters=512, kernel size=4, strides=1, padding=no, activation=relu).
- *Dropout Layer* (dropout=0.5).
- *1D CNN Layer* (filters=256, kernel size=4, strides=1, padding=no, activation=relu).
- *Primary Capsule Layer* (capsule dim=8, channels=32, kernel size=4, strides=2, padding=no).
- *Capsule Layer* (capsules=1, capsule dim=16, routing=7).
- *Length Layer*.

3.2.3. Hybrid models

CNN-LSTM. The CNN and LSTM hybrid model first uses a CNN layer where the convolution operations extract the feature sequence from the input (the embedded vector) and then passes the feature sequence to an LSTM layer. The architecture of CNN-LSTM is:

- *Embedding Layer* (embeddings dim=50).
- *1D CNN Layer* (filters=256, kernel size=4, strides=1, padding=same, activation=relu).
- *LSTM Layer* (unit=128, activation=relu).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

CNN-Bi-LSTM. The CNN and Bi-LSTM hybrid model replaces the LSTM layer in CNN-LSTM with the bidirectional LSTM layer. The architecture of CNN-Bi-LSTM is:

Table 1
Time complexity*.

FANCI	RNN	Bi-RNN	LSTM
$O(N \log(N) ID)$	$O\left(N\left(IH + H^2 + HK\right)\right)$	$O\left(N\left(IH + H^2 + HK\right)\right)$	$O\left(N\left(4IH + 4H^2 + 3H + HK\right)\right)$
Bi-LSTM	**GRU**	**Bi-GRU**	**Stack-LSTM**
$O\left(N\left(4IH + 4H^2 + 3H + HK\right)\right)$	$O\left(N\left(3IH + 3H^2 + HK\right)\right)$	$O\left(N\left(3IH + 3H^2 + HK\right)\right)$	$O\left(N\left(4IH + 8H^2 + 6H + HK\right)\right)$
Stack-Bi-LSTM	**Bi-LSTM-Att**	**CNN**	**CNN-Maxpool**
$O\left(N\left(4IH + 4H^2 + 3H + HK\right)\right)$	$O\left(N\left(4IH + 4H^2 + 4H + HK\right)\right)$	$O\left(N \cdot n_0 \cdot s_1^2 \cdot n_1 \cdot m_1^2\right)$	$O\left(N \cdot n_0 \cdot s_1^2 \cdot n_1 \cdot m_1^2\right)$
CNN-Avepool	**Stack-CNN-Maxpool**	**CoreDNS-CNN**	**Par-CNN**
$O\left(N \cdot n_0 \cdot s_1^2 \cdot n_1 \cdot m_1^2\right)$	$O\left(N \cdot \sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right)$	$O\left(N \cdot n_0 \cdot s_1^2 \cdot n_1 \cdot m_1^2\right)$	$O\left(N \cdot \max_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right)$
CapsNet	**CNN-LSTM**	**CNN-Bi-LSTM**	**CNN-Bi-LSTM-Att**
$O\left(N \cdot \sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2\right)$	$O(N(n_0 \cdot s_1^2 \cdot n_1 \cdot m_1^2 + 4m_1H + 4H^2 + 3H + HK))$	$O(N(n_0 \cdot s_1^2 \cdot n_1 \cdot m_1^2 + 8m_1H + 8H^2 + 6H + 2HK))$	$O(N(n_0 \cdot s_1^2 \cdot n_1 \cdot m_1^2 + 8m_1H + 8H^2 + 8H + 2HK))$

* N: number of training samples; I: dimension of input data; D: number of decision trees; H: number of hidden units: K: dimension of output data; l of n_l, s_l, and m_l: index of a convolutional layer; d: number of convolutional layers; n_l : number of filters in the l-th convolutional layer; $n_0 = I$: number of input channels of the 1-th layer; s_l: size of the filter in the l-th convolutional layer; m_l: size of the output feature map in the l-th convolutional layer.

- *Embedding Layer* (embeddings dim=50).
- *1D CNN Layer* (filters=256, kernel size=4, strides=1, padding=same, activation=relu).
- *Bidirectional LSTM Layer* (unit=128, activation=relu).
- *Dropout Layer* (dropout=0.5).
- *Dense Layer* (unit=1, activation=sigmoid).

CNN-Bi-LSTM-Att. We enhance CNN-Bi-LSTM by adding the attention mechanism to its bidirectional LSTM layer and thereby obtain the CNN and bidirectional LSTM hybrid model with the attention mechanism. The architecture of CNN-Bi-LSTM-Att is:

- *Embedding Layer* (embeddings dim=50).
- *1D CNN Layer* (filters=256, kernel size=4, strides=1, padding=same, activation=relu).
- *Bidirectional LSTM Layer* (unit=128, activation=tanh).
- *Attention Layer*
- *Dropout Layer* (dropout=0.2).
- *Dense Layer* (unit=1, activation=sigmoid).

4. Evaluation

4.1. Time complexity

The time complexity of all models are presented in Table 1.

Table 2
Confusion matrix of classification.

Actual Class \ Predicted Class	Malicious	Benign
Malicious	True Positive (TP)	False Negative (FN)
Benign	False Positive (FP)	True Negative (TN)

4.2. Datasets

We use the datasets in our evaluation from two sources:

- The domain names from DomCop [21] are used as the dataset of benign domain names. The dataset contains the top 10 million domains taken from the Open PageRank Initiative. The PageRanks are calculated based on the Open data provided by Common Crawl and Common Search.
- The COVID-19 Cyber Threat Coalition Blocklist for malicious domain names [22] is used as the dataset of COVID-19 related domain names. The data set was published with indicators to be used by criminals trying to prey on individuals, organizations, businesses, and governments using the COVID-19 pandemic. It contains 93,140 domain names. One may think of identifying the malicious domain names using the COVID-19 themed keywords like "COVID", "pandemic", "coronavirus", etc. However, that tactic is unlikely to work because an overwhelming majority of the dataset does not contain such keywords.

4.3. Experiment setting

We created 100 benign sets and 100 abusing sets by randomly sampling with replacement 100 times from the DomCop set and the COVID-19 Cyber Threat Coalition Blocklist set stated above respectively. Each sampled set contains 10,000 domains for both the benign sets and the abusing sets. For each pair of benign set and malicious set, stratified 5-fold cross-validation is performed. We perform 500 experiments given the 5-fold computation over the 100 pairs of datasets. For each experiment, we calculate the evaluation metrics specified in Section 4.4 and use the average over the 500 experiments as the overall measure of detection rate. For each experiment, we also record the computing time for both training and inference and use the average over the 500 experiments as the overall measure of computing time.

We preprocess the original domain names by truncating a long domain name or padding a short domain name to a fixed length of characters. In our pre-processing, we truncate or pad each domain name to a fixed length. We set the fixed length to 256 for CoreDNS-CNN (then reshaped to a 2D matrix) and 50 for the other models. The valid characters for a domain name include letters (i.e., a-z, A-Z), numbers (i.e. 0-9), dot(.), and hyphen (-). A padding character should also be included for our machine learning task. Thus overall, the number of valid characters is 39. We use the Adam algorithm [23] whose learning rate is set to 0.001 for our deep learning models. And we configure a batch size of 256 in our mini-batch training. The detection accuracy is monitored during training and the training is terminated when the detection accuracy is not improved over 10 consecutive epochs. The model with the best detection accuracy in the training history is selected as the best model.

All models were implemented in Python 3.7. And all deep learning models were implemented using Tensorflow-GPU version 2.0. We evaluate the models on the NVIDIA DGX Station only using one GPU of the four.

4.4. Evaluation metrics

4.4.1. Detection rate metrics

To evaluate how good a binary classifier is in predicting a class of a sample, we use the following four detection rate metrics: accuracy, precision, recall, and F1, whose definitions are based on the confusion matrix of classification in Table 2.

Accuracy (ACC) is the proportion of the correct predictions among the total predictions:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN} \tag{15}$$

Precision (PRE) is the proportion of the true positives among the predicted positives:

$$PRE = \frac{TP}{TP + FP} \tag{16}$$

Recall (REC) is the proportion of the predicted positives among the actual positives:

$$REC = \frac{TP}{FN + TP} \tag{17}$$

F1 is calculated as the harmonic mean of precision and recall:

$$F1 = 2 \times \frac{PRE \times REC}{PRE + REC} \tag{18}$$

4.4.2. Computing time metrics

As we apply mini-batch gradient descent to the training of the deep learning models, one experiment may cost more than one epochs before the stopping criterion is met. Specifically, no experiment should take less than 11 epochs as the stopping criterion of successive non-improvement is set to 10 epochs. So we use the overall training time divided by the size of the training set as one training time metric which is averaged over all training samples and all training epochs.

The overall training time is correlated to both the number of epochs and the training time per epoch. So we use the training time per epoch divided by the size of the training set as the other training time metric which is averaged over all training samples, all training batches, and all training epochs. Note that the overall training time may be heavily impacted by the learning rate of the optimization algorithm while the training time per epoch is not. As a conventional machine learning model does not take multiple epochs, the training time per epoch metric does not apply to it.

To measure the prediction time, we use the prediction time divided by the size of the validation set and thus the metric is averaged over all validation samples.

4.5. Results and analysis

Table 3 shows the results under the evaluation metrics for all model implementations.

Table 3: Results.

Model	Accuracy	Precision	Recall	F1	Training Time[a,b]	Prediction Time[b]
FANCI	0.9215	0.9219	0.9215	0.9215	1.4645	**0.0284**
RNN	0.9785	0.9787	0.9785	0.9785	6.7335 0.1415	0.5151
Bi-RNN	0.9800	0.9802	0.9800	0.9800	15.3784 0.2207	1.0811
LSTM	0.9860	0.9862	0.9860	0.9860	12.2777 0.3196	0.7919
Bi-LSTM	0.9865	0.9869	0.9865	0.9865	35.0317 0.5518	1.2960
GRU	0.9855	0.9858	0.9855	0.9855	8.2590 0.2902	0.7206
Bi-GRU	0.9865	0.9868	0.9865	0.9865	11.9962 0.4717	1.1434
Stack-LSTM	0.9865	0.9868	0.9865	0.9865	25.8137 0.6089	1.4990
Stack-Bi-LSTM	0.9865	0.9867	0.9865	0.9865	43.4169 1.1010	2.5168
Bi-LSTM-Att	0.9855	0.9858	0.9855	0.9855	4.1881 0.1350	1.6950

Table 2: Results (continued).

Model	Accuracy	Precision	Recall	F1	Training Time[a,b]	Prediction Time[b]
CNN	0.9810	0.9811	0.9810	0.9810	1.5480 0.0776	0.0992
CNN-Maxpool	0.9850	0.9852	0.9850	0.9850	2.5896 0.0902	0.1356
CNN-Avepool	0.9850	0.9851	0.9850	0.9850	2.5905 0.0909	0.1362
Stack-CNN-Maxpool	0.9860	0.9861	0.9860	0.9860	3.0361 0.1213	0.1835
CoreDNS-CNN	0.7991	0.8129	0.7992	0.7969	**0.9755** **0.0363**	0.0878
Par-CNN	0.9855	0.9858	0.9855	0.9855	3.7357 0.1256	0.2354
CapsNet	0.9850	0.9852	0.9850	0.9850	31.0569 1.0544	0.7887
CNN-LSTM	**0.9870**	**0.9873**	**0.9870**	**0.9870**	54.6994 1.7051	3.9395
CNN-Bi-LSTM	**0.9870**	**0.9873**	**0.9870**	**0.9870**	71.8739 2.8665	6.2094
CNN-Bi-LSTM-Att	0.9850	0.9852	0.9850	0.9850	3.9693 0.1451	2.1574

[a] The upper value is the overall training time and the lower value is the training time per epoch for the deep learning models.
[b] in ms.

4.5.1. Detection rate

In general, the deep learning models demonstrate better detection rates than the conventional machine learning model, namely FANCI. By examining the feature set, we can find that FANCI processes some ordering-related information in its feature design. For example, the ratios of repeated characters, consecutive consonants, and consecutive digits are used to capture deviations from common linguistic patterns of domain names. However, lots of sequence information cannot be fully represented by the FANCI features. And deep learning models gain better semantic insights by constructing models that directly embed the sequencing information into themselves.

We also identify that the only exception of deep learning models over FANCI is CoreDNS-CNN which has a lower accuracy of 0.7991. CoreDNS-CNN differs from the other deep learning models in two ways: it does not use the 1D convolution but instead uses the 2D convolution over the reshaped domain name string; it does not use embeddings. Unfortunately, the 2D reshaping does not conform to the nature of the domain name string. For instance, we don't expect the 257th character should be better related to the 1st character than the 256th character while they are closer in the 2D matrix after the reshaping. On the other hand, embeddings should be essential for modeling quasi-natural language sequences like domain names. Neural network embeddings are useful because they can reduce the dimensionality of categorical variables and meaningfully represent categories in the transformed space.

RNN, Bi-RNN, and CNN (without pooling) are less effective than the other deep learning models, except CoreDNS-CNN, which have relatively similar performances with an accuracy between 0.9850 and 0.9870.

By comparing the RNN class, namely RNN, LSTM, and GRU as well as their bidirectional versions, we can find that $LSTM > GRU > RNN$ and $bidirectional\ version > unidirectional\ version$ in terms of detection rate. Recalling the architectures introduced in Section 2.2.1, LSTM and GRU both add gates to RNN and therefore can pass more relevant information down the long chain of sequences to make predictions, and the key difference between GRU and LSTM is that a GRU unit has two gates (reset and update gates) whereas an LSTM unit has three gates (namely input, output and forget gates). So moving from RNN to GRU, and then LSTM gives us the most control over the flow of information and thus, better detection rate, but also comes with more complexity and operating cost. Generally, a bidirectional model shows better detection rate than its unidirectional counterpart despite that the margin may vary across different RNNs. That is because a bidirectional model uses two hidden states combined at any time step to

preserve information from both past and future and thereby can understand the context better than its unidirectional counterpart which only relies on information from the past.

The simple CNN has only one shallow convolutional layer while the other CNN models have either pooling or stacked convolutional layers. The convolution, pooling, and ReLU layers are typically interleaved in a CNN model in order to increase the expressive power of the neural network, and more complex sequences generally call for deeper networks that enable the hierarchical features. Not having enough layers effectively prevents the network from learning the hierarchical regularities in the sequence that are combined to create its semantically relevant components. Therefore the result may indicate that the complexity of the simple CNN may underfit the learning task.

We can see that stacking LSTM or Bi-LSTM does not improve the detection rate of a single-layer LSTM or Bi-LSTM. Moreover, if the model gets more complex by adding the attention mechanism to Bi-LSTM, the detection rate is reduced due to overfitting. That suggests that Bi-LSTM is a relatively good match with the learning problem with less underfitting and overfitting issues among the RNN class.

For CNN with pooling, max pooling and average pooling demonstrate almost the same detection rate. It seems that the choice of pooling type has little impact on the detection rate.

CNN-LSTM and CNN-Bi-LSTM achieve the best detection rate among all models. Given the extra overhead introduced by moving CNN-LSTM to CNN-Bi-LSTM, CNN-LSTM is shown to be the best model in terms of detection rate. Adding convolutional layers to capture local and temporal patterns on top of LSTM layers is helpful in this scenario.

It looks like the attention mechanism does not help in the learning task. That is because the length of the sequence is not long enough to let the attention mechanism capture the long-term dependency. As a result cross to our purpose, the adverse effect of introducing the attention mechanism is overfitting.

4.5.2. Computing time

FANCI is the fastest model in prediction and CoreDNS-CNN is the fastest model in training. As an RF-based classifier is chosen as optimal, FANCI's prediction speed is largely dependent on the depth of the decision tree and the number of features, both of which determine the lighter weight of computation compared with the neural networks. CoreDNS-CNN differs from other deep learning models in that it does not use an embedding layer as a front end. The results show that omitting an embedding layer can save significant training time at the non-trivial cost of a lower detection rate.

The CNN class models generally show better training speed and prediction speed than the RNN class models. That is because CNNs are faster by design since the computations in CNNs can run in parallel (the same filter applied to multiple locations of the sequence at the same time), while RNNs need to be processed sequentially since the subsequent steps depend on previous ones.

CNN-LSTM and CNN-Bi-LSTM are the least competitive in terms of both training time and prediction time despite their advantages in the detection rate. There is a non-trivial increase in both training time and prediction time by moving a unidirectional RNN to its bidirectional counterpart.

4.5.3. Overall remarks

There should be some tradeoff between the detection rate metrics and the computing time metrics because no models excel simultaneously in both metrics: CNN-LSTM has the best detection rate but a less competitive computing time; FANCI has the best prediction time and a good training time but a less competitive detection rate; CoreDNS-CNN has the best training time and a good prediction time but the worst detection rate.

If the detection rate is the overwhelming top priority, CNN-LSTM may be a good choice. If a balance between the detection rate and the computing time is desired, CNN with pooling and stacked CNN with pooling may be good candidates that have some detection rates comparable to CNN-LSTM and save over 93% computing time of CNN-LSTM.

4.6. High-Frequency Lexical Analysis

We conduct high-frequency lexical analysis on the COVID-19 related domain name list as well as the benign domain name list. First, the substrings of each domain name string, with 3 or more characters, are extracted. Second, the domain names containing at least one of each substring S are counted for S. Third, all substrings are ordered by their counts. Consider that a high-frequency substring S may be a substring of another high-frequency substring S^+. For example, if "example.com" is a high-frequency substring, its substrings like "example", "exam", "com", etc.

Table 4
High-frequency substrings.

Substring	Count	Information
dot-millinium.ey.r.appspot	5784	Some domain names containing this substring were labeled as malicious[a] or specifically risk phishing.[b]
uc.r.appspot	1713	Some domain names containing this substring were labeled as malicious.[c] In particular, the domain name "covid-19-safedating.uc.r.appspot.com"[d] indicates the close relation to COVID-19.
gleowayel400503.uc.r.appspot	1591	Some domain names containing this substring were labeled as risk phishing.[e]
heartchakracheckup	583	The heart chakra is the fourth primary chakra, according to Hindu Yogic, Shakta and Buddhist Tantric traditions. The heart chakra governs one's senses of trust, fearlessness, peace, generosity, gratitude, and connectedness, as well as change and transformation, healthy boundaries, depth in relationships with others, emotional control, and love for oneself.[f] So it is related to health and thereby COVID-19.

[a] https://www.hybrid-analysis.com/sample/c3724edf742232117e142f0665a336f33cb2c8f93657766a693e5e470cdee2e5/6104b600d390aa665c6855d9
[b] https://www.riskiq.com/lookup/dkvcfmbcumexncut-dot-millinium.ey.r.appspot.com
[c] https://www.joesandbox.com/analysis/230388/0/html
[d] https://www.riskiq.com/lookup/covid-19-safedating.uc.r.appspot.com
[e] https://www.riskiq.com/lookup/qsynsrkdhsfyaolgkcrlwgsyjh-dot-gleowayel400503.uc.r.appspot.com
[f] https://en.wikipedia.org/wiki/Anahata

should have no less frequency than "example.com". So a long substring with a maximum length is used to replace all its substrings in the sorted high-frequency substring list. Using this merging approach, we can obtain the maximum length substring list for the high-frequency substring list. By comparing the maximum length substring lists for the COVID-19 related and benign domain name lists, we find some substrings whose occurrences are highly frequent for the COVID-19 related domain names but not for the benign domain names. These substrings can be considered closely related to COVID-19 and highly suspicious. A few examples of these substrings are listed in Table 4 where we also provide their counts and some additional information regarding security risks and relevance to COVID-19. They are related to security risks or COVID-19 or both.

5. Discussion

5.1. Limitations

One major limitation of our proposed supervised machine learning approaches is a reliance on labeled data. When training a supervised machine learning model, e.g., a classifier in this work, we need ground-truth labels given to sufficient samples of each class. However, providing labels for datasets is either very expensive or not possible at all in many cases. Even if labeled data are given, incomplete, inaccurate, or inexact labels can also greatly impact supervised machine learning performance. The other limitation is the non-trivial computation time of supervised machine learning. To speed up training and prediction, high computation power is often needed especially for real-time applications.

5.2. Implications

While this work focuses on detecting COVID-19 related domain names, the proposed detection models are also applicable to general abnormal/malicious domain name detection tasks. DNS, which translates IP addresses into hostnames and visa versa, is a critical component of the Internet. First, DNS is relied on by almost every online application, with malware and malicious actors included. Malware commonly needs to use DNS to receive instructions, exfiltrate data, or otherwise communicate with attackers. Second, DNS becomes a highly attractive target for threat actors because of its ubiquity. For example, disrupting some vital DNS services may result in a devastating collapse on the large-scale Internet, and hacking some DNS data may compromise the entire business of the relevant enterprises. As the recent Facebook outage in October 2021 showed, failing DNS was the first symptom found in diagnosing the

cause. Given the considerations above, DNS is a good source of information for security analysis and threat prediction problems like monitoring cybercrimes at the state level, in situations of hostilities, during riots, or even when tracking the planning of terrorist attacks. As a basic DNS feature, the domain name can be considered as a threat/abnormality indicator. The supervised learning models used in this work can automatically distinguish malicious/abnormal domain names from benign/normal ones as long as they should demonstrate different characteristics.

Our proposed machine learning models, especially the deep neural network architectures, can be used to address general abnormal/malicious domain name detection tasks. Thinking about the training of a machine learning model, there is no fundamental difference between the COVID-19 related domain name detection and other malicious domain name detection problems because they both use a domain name string itself as input and a prediction label as output. In addressing a binary classification task, the deep neural network models can automatically capture the characteristics of the two-class labeled training set and use the characteristics to predict the label of a domain name. That is, the models can easily adapt to different training data for different specific abnormal/malicious domain name detection tasks. In a broader sense, if we think of a domain name as a short text, our proposals can be considered as general solutions to a short text detection problem and potentially have applications in other similar contexts. Thus even if the COVID-19 pandemic should end or transition to normalcy sometime in the future, the work is still relevant to other applications. Nevertheless, this work aims at providing a direct timely solution to the COVID-19 related domain name detection problem. To this end, the machine learning models were fully optimized and evaluated for the specific task for COVID-19 domain name detection in order to offer guidance in determining the optimal domain name detection model to use in real-world deployments. When adapting to a different specific task, the models may need some fine-tuning of hyperparameters and/or the neural network architecture as part of optimization efforts. Additionally, an extensive evaluation similar to what was done for the task in this work is recommended as models may perform differently across different specific tasks. As for hyperparameter tuning which searches for the best configuration of hyperparameters to enable optimal performance, we can adopt grid search, random search, evolutionary algorithms, Bayesian optimization, etc. When applied to other specific tasks, the hyperparameter tuning can rely on the hyperparameter set specified in this work as the starting point of search. Grid search performs an exhaustive search in the hyperparameter search space. Random search performs a randomized search over hyperparameters from certain distributions. Evolutionary algorithms work by modifying a set of candidate solutions according to certain rules. Bayesian optimization builds a surrogate probabilistic model to map hyperparameters to the objective function, finds the hyperparameters that perform best over the surrogate model, evaluates the best hyperparameter configuration, and updates the surrogate model based on the new evaluation, and moves towards the optimum by iterating the process above. Neural network architecture tuning, as a subset of hyperparameter tuning, is specialized for optimizing neural network architectures. Neural network architecture tuning introduces significant algorithmic and computational complexities in comparison with hyperparameter optimization. While neural network architecture tuning is still significantly challenging, some remarkable advances [24, 25] have been achieved in recent years which can apply to the neural network architecture adaptation to a new specific task.

5.3. Considerations

Using more extensive data than domain name data is generally more likely to boost detection performance. For example, some previous approaches rely on network traffic data which are considered more informative than pure domain names. However, a lack of sufficient data resources has become a bottleneck to cyber security practitioners these days. Extensive data like network traffic data are either very costly to obtain or inaccessible at all. So we consider domain name strings themselves as the minimum information we can have when determining malicious domain names. The detection techniques using pure domain names can greatly ease their adoption in the real world, especially for those who do not have extensive data. In case that extensive data are available, the detection techniques using pure domain names can also be applied as part of a larger and more complex detection system. For example, a detection model using pure domain names and a detection model using other network traffic features can be combined using some "stacking" approaches for better detection performance. One most common "stacking" approach is voting, where the prediction probabilities given by each model for each class are summed and the label with the largest summed probabilities is selected. Even if used as a component in a larger detection system, our efforts in this work still count because improving one model among the "stacked" models will generally contribute to the overall prediction performance.

6. Conclusions

In this paper, we empirically evaluated supervised machine learning models for the COVID-19 related domain name detection problem. Our evaluation included both conventional machine learning models and deep learning models with three architectures: recurrent, convolutional, and hybrid models. Detection rate metrics and computing time metrics were both evaluated. The evaluation clearly demonstrated the superiority of the deep learning models over the conventional machine learning models, especially in terms of detection rate. There was evidence that some tradeoff between detection rate and computing speed is needed for the choice of deep learning models. Considering the tradeoff, CNN with pooling and stacked CNN with pooling tended to strike a better balance despite that CNN-LSTM showed the best detection rate.

References

[1] Palo Alto Networks, "Studying How Cybercriminals Prey on the COVID-19 Pandemic," unit42.paloaltonetworks.com/how-cybercriminals-prey-on-the-covid-19-pandemic/. Accessd Nov 15, 2020.

[2] S. Schüppen, D. Teubert, P. Herrmann, and U. Meyer, "FANCI: Feature-based automated NXDomain classification and intelligence," in *Proceedings of the 27th USENIX Security Symposium*, 2018, pp. 1165–1181.

[3] Y. Qiao, B. Zhang, W. Zhang, A. K. Sangaiah, and H. Wu, "DGA domain name classification method based on Long Short-Term Memory with attention mechanism," *Applied Sciences (Switzerland)*, vol. 9, no. 20, 2019.

[4] A. D. Kumar, H. Thodupunoori, R. Vinayakumar, K. P. Soman, P. Poornachandran, M. Alazab, and S. Venkatraman, "Enhanced domain generating algorithm detection based on deep neural networks," in *Advanced Sciences and Technologies for Security Applications*, 2019, pp. 151–173.

[5] S. Zhou, L. Lin, J. Yuan, F. Wang, Z. Ling, and J. Cui, "CNN-based DGA detection with high coverage," in *2019 IEEE International Conference on Intelligence and Security Informatics, ISI 2019*, 2019, pp. 62–67.

[6] F. Ren, Z. Jiang, X. Wang, and J. Liu, "A DGA domain names detection modeling method based on integrating an attention mechanism and deep neural network," *Cybersecurity*, vol. 3, no. 1, 2020.

[7] P. Vij, S. Nikam, and A. Bhatia, "Detection of Algorithmically Generated Domain Names using LSTM," in *2020 International Conference on COMmunication Systems and NETworkS, COMSNETS 2020*, 2020, pp. 1–6.

[8] B. Liu and H. Wang, "Real-time monitoring system for dga domain based on long short-term memory," in *Advances in Intelligent Systems and Computing*, vol. 1244 AISC, 2021, pp. 159–165.

[9] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting Domain Generation Algorithms with Long Short-Term Memory Networks," *arXiv preprint arXiv:1611.00791*, 2016.

[10] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[11] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the Properties of Neural Machine Translation: Encoder–Decoder Approaches," 2015, pp. 103–111.

[13] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[14] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[15] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, vol. 2017-December, 2017, pp. 3857–3867.

[16] B. Dhingra, Z. Zhou, D. Fitzpatrick, M. Muehl, and W. W. Cohen, "Tweet2Vec: Character-based distributed representations for social media," *54th Annual Meeting of the Association for Computational Linguistics, ACL 2016 - Short Papers*, pp. 269–274, 2016.

[17] X. Zhang, J. Zhao, and Y. Lecun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems*, vol. 2015-January, 2015, pp. 649–657.

[18] C. Ekbote, "CoreDNS-CNN," github.com/cekbote/coredns_ml_plugin. Accessed Nov 15, 2020.

[19] D. S. Berman, "DGA CapsNet: 1D application of capsule networks to DGA detection," *Information (Switzerland)*, vol. 10, no. 5, 2019.

[20] S. Vosoughi, P. Vijayaraghavan, and D. Roy, "Tweet2Vec: Learning tweet embeddings using character-level CNN-LSTM encoder-decoder," in *SIGIR 2016 - Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2016, pp. 1041–1044.

[21] DomCop, "Benign domain name list," www.domcop.com/top-10-million-domains. Accessd Nov 15, 2020.

[22] COVID-19 Cyber Threat Coalition, "COVID-19 related domain name List," blacklist.cyberthreatcoalition.org/vetted/domain.txt. Accessd Nov 15, 2020.

[23] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[24] C. Termritthikun, Y. Jamtsho, J. Ieamsaard, P. Muneesawang, and I. Lee, "EEEA-Net: An Early Exit Evolutionary Neural Architecture Search," *Engineering Applications of Artificial Intelligence*, vol. 104, p. 104397, 2021.

[25] Y. Li, C. Hao, P. Li, J. Xiong, and D. Chen, "Generic Neural Architecture Search via Regression," in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

Authors' Biographies

Zheng Wang is with the Internet Technologies Research group at the National Institute of Standards and Technology (NIST). His research interests include machine learning and its applications to networking and communication problems, network measurement, and distributed systems.