

Article

A Payload Based Malicious HTTP Traffic Detection Method Using Transfer Semi-Supervised Learning

Tieming Chen, Yunpeng Chen, Mingqi Lv *, Gongxun He, Tiantian Zhu, Ting Wang  and Zhengqiu Weng

College of Computer Science, Zhejiang University of Technology, Hangzhou 310023, China; tmchen@zjut.edu.cn (T.C.); tuertong99@gmail.com (Y.C.); gongxunh@zjut.edu.cn (G.H.); ttzhu@zjut.edu.cn (T.Z.); wangting@zjut.edu.cn (T.W.); derisweng@wzpt.edu.cn (Z.W.)

* Correspondence: mingqilv@zjut.edu.cn; Tel.: +86-1366-6600-887

Abstract: Malicious HTTP traffic detection plays an important role in web application security. Most existing work applies machine learning and deep learning techniques to build the malicious HTTP traffic detection model. However, they still suffer from the problems of huge training data collection cost and low cross-dataset generalization ability. Aiming at these problems, this paper proposes DeepPTSD, a deep learning method for payload based malicious HTTP traffic detection. First, it treats the malicious HTTP traffic detection as a text classification problem and trains the initial detection model using TextCNN on a public dataset, and then adapts the initial detection model to the target dataset based on a transfer learning algorithm. Second, in the transfer learning procedure, it uses a semi-supervised learning algorithm to accomplish the model adaptation task. The semi-supervised learning algorithm enhances the target dataset based on a HTTP payload data augmentation mechanism to exploit both the labeled and unlabeled data. We evaluate DeepPTSD on two real HTTP traffic datasets. The results show that DeepPTSD has competitive performance under the small data condition.



Citation: Chen, T.; Chen, Y.; Lv, M.; He, G.; Zhu, T.; Wang, T.; Weng, Z. A Payload Based Malicious HTTP Traffic Detection Method Using Transfer Semi-Supervised Learning. *Appl. Sci.* **2021**, *11*, 7188. <https://doi.org/10.3390/app11167188>

Academic Editor: Andrea Prati

Received: 26 June 2021

Accepted: 29 July 2021

Published: 4 August 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: malicious traffic detection; HTTP payload; Data augmentation; Transfer learning; semi-supervised learning

1. Introduction

Web applications are the most popular applications on Internet, as they require a smaller amount of client side resources, and are relaxed to deliver and maintain. Therefore, a large number of attackers put web applications as their primary target. Among all the network protocols, HTTP (Hypertext Transfer Protocol) is used by most of the web applications for communication between web browsers and web servers, and a large majority of attacks against web applications are launched by using malicious HTTP requests, e.g., SQL injection, XSS (Cross-Site Scripting). Therefore, malicious HTTP traffic detection systems play an essential role in establishing a secure and reliable web application. In practice, the malicious HTTP traffic detection models are usually deployed on a WAF (Web Application Firewall) in front of the web servers.

The existing malicious network traffic detection methods could be divided into two categories according to the exploited information, i.e., statistic based methods and payload based methods. Statistic based methods [1–5] extract statistical features (e.g., duration of flow, inter arrival time, number of packets, etc.) from network traffic analyzer (e.g., NetFlow [6]), and then apply rule based or learning based techniques to detect malicious network traffic. These statistical features are payload-independent, and thus the statistic based methods can address specific issues such as data privacy and traffic encryption [7]. However, statistic based methods have the following limitations. First, useful information such as suspicious keywords in payloads are ignored, so it is difficult to detect malicious network traffic that does not exhibit significant differences in flow patterns (e.g., SQL injection). Second, most statistical features require a relative long period of monitoring time

before they can be appropriately extracted, and thus they are usually used for representing network traffic based on protocols such as TCP and UDP, and not suitable for malicious network traffic detection from single HTTP requests. On the other hand, payload based methods [8–15] detects malicious network traffic by analyzing the content of the packets (e.g., URL, file path, request body, etc.), and thus they can be used for single HTTP request. Therefore, these two types of detection methods can be used to complement each other. In this paper, we focus on the payload based malicious HTTP traffic detection method.

Most state-of-the-art payload based methods adopt learning techniques for malicious network traffic detection, including traditional machine learning techniques [8–11] and deep learning techniques [12–16]. They train detection models by exploiting a corpus of correctly labeled samples (including both normal and various types of malicious network traffic samples), and then utilize the detection models for real-time malicious network traffic detection. Although learning based methods could achieve high detection accuracy, they still suffer from the following problem: the learning techniques (especially the deep learning techniques) require a large number of labeled samples to train the models. However, it is very difficult to collect adequate labeled samples in practice, especially the malicious network traffic samples. Although this problem could be alleviated by training the detection models on a public dataset with a relatively large number of labeled samples and then applying the trained models to the target systems, there is a significant pattern shift between the samples collected from different systems (called the train-test gap problem), due to the unique characteristic of each individual system [17]. Applying the models trained on the public dataset to a new system usually results in performance degradation.

To address the above problems, we propose DeepPTSD, a payload based malicious HTTP traffic detection method based on deep transfer semi-supervised learning technique. The main contributions of this paper are summarized as follows.

(1) We propose a deep learning framework for malicious HTTP traffic detection. To ensure high generalization ability under the condition that the target system has very limited number of labeled samples, we leverage the labeled samples of a public dataset to train the initial detection model, and then fine-tune the initial detection model to adapt to the target system based on a transfer semi-supervised learning model. To the best of our knowledge, this is the first solution of combining deep learning with transfer learning and semi-supervised learning for by augmenting the payload dataset malicious HTTP traffic detection.

(2) We propose a HTTP payload data augmentation method based on a novel keyword avoidance perturbation mechanism. To increase the diversity of the training samples, we generate pseudo labeled samples by imposing noises to the original labeled HTTP traffic samples while keeping the key portion unchanged. The pseudo labeled and original labeled samples are then utilized collaboratively in the semi-supervised learning algorithm.

(3) We conduct cross-dataset validation on two real datasets to evaluate DeepPTSD. The results show that DeepPTSD has competitive performance under the small data condition.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 details the proposed method. Section 4 reports the experiment results. Section 5 concludes the paper.

2. Related work

2.1. Malicious Network Traffic Detection

The existing malicious network traffic detection methods could be divided into two categories according to the exploited information, i.e., statistic based methods and payload based methods. Statistic based methods extract flow based statistical features (e.g., duration of flow, inter arrival time, number of packets, etc.) and apply rule based or machine learning based techniques to detect malicious network traffic [1–5]. Although statistic based methods could adapt to data privacy and encryption, they cannot detect malicious network traffic that does not exhibit significant differences in flow patterns (e.g., SQL injection). In addition, they require a relative long period of monitoring time to extract

statistical features, so they are usually designed for network traffic based on protocols such as TCP and UDP, and not suitable for malicious network traffic detection from single HTTP requests. Payload based methods detect malicious network traffic by exploiting the content of the packets (e.g., URL, file path, request body, etc.). Most state-of-the-art payload based methods adopt traditional machine learning techniques and deep learning techniques for malicious network traffic detection. Given a set of network traffic samples as training data, learning based methods train a detection model, which could classify a new network traffic sample as malicious or benign. The key factor for the learning based methods is to extract good features to represent the network traffic samples. Traditional machine learning techniques explore features from various aspects, including lexical features (e.g., N-gram features, bag-of-words features, etc.) [8,9,18,19] and statistical semantic features (e.g., number of characters, number of parameters, count of sensitive keywords, etc.) [10,11,20]. However, these features should be manually designed based on domain knowledge and require constant adjustment to accommodate changes in malicious network traffic patterns.

In recent years, thanks to the superior capability of automatically extracting representative features from raw data (especially the unstructured data), deep learning techniques have been exploited for malicious network traffic detection [16]. For example, Saxe et al. [21] proposed eXpose, which is a malicious payload detection model based on character-level embedding and CNN (convolutional neural network). Park et al. [12] proposed a malicious HTTP message detection model by using an autoencoder with character-level binary image transformation. Peng et al. [13] proposed a joint approach of CNN, LSTM, and attention mechanism to detect malicious HTTP traffic by exploiting character-level lexical features and host features. However, character-level features cannot capture the latent semantics of payloads, resulting in low generalization ability. Aiming at this problem, more recent researches design detection model based on word-level features. For example, Yu et al. [22] modeled a HTTP traffic sample as a natural language sequence under predefined vocabulary using BiLSTM (bidirectional long short-term memory) with attention mechanism. Le et al. [14] proposed URLNet, an end-to-end deep learning framework to learn a non-linear URL embedding by applying CNN to both characters and words. Yang et al. [15] designed a convolutional GRU (gated recurrent unit) for malicious URL detection based on a predefined keyword library. However, deep learning techniques require a large number of labeled samples to train the models, while it is almost impossible to collect adequate labeled samples for each individual system, especially the malicious samples.

2.2. Small Data Learning

Small data learning is referred to the techniques designed to carry out learning on the condition of lacking sufficient training samples [23], and the main strategies of small data learning include data augmentation and knowledge transfer.

The data augmentation strategy attempts to compensate the training data by adding slightly modified copies of the existing data or newly created synthetic data from the existing data, with the purpose of reducing overfitting when training a learner [24–26]. On the other hand, knowledge transfer strategy tries to train a learner for the target domain by leveraging knowledge from other domains, and could be divided into three categories according to the types of transferred knowledge, i.e., representation transfer, model transfer, and cognition knowledge transfer. Representation transfer attempts to transfer the knowledge of representing the same object from different domains [27]. Model transfer uses the learner trained from other domains to fit the small training data in the target domain through fine-tuning its parameters [28,29]. Cognition knowledge transfer leverages high-level knowledge such as common sense, domain knowledge, and other concepts [30,31].

Small data learning is widely used in research areas such as computer vision and natural language processing. In contrast, the application of small data learning in malicious network traffic detection has been much more limited, especially when combined with

deep learning. In our work, we propose a novel data augmentation scheme for payload data, and design a small data learning framework based on the data augmentation scheme.

3. Methodology

3.1. Preliminary

First, we elaborate the data used for payload based malicious HTTP traffic detection. Malicious information of web attacks is usually contained in the content of the request URL (if a GET request is sent) or the request body (if a POST request is sent). If a HTTP traffic sample is a GET request, its payload is the path of the URL. If a HTTP traffic sample is a POST request, its payload is the path of the URL and the body of the request (i.e., the request parameters). To this end, a HTTP traffic sample s is defined as a tuple $s = (x, y)$ where x denotes the payload extracted from s (in the form of a string), and y denotes the label of s (i.e., normal or the malicious type). Note that y is unknown if s is a testing sample. We give an example of HTTP traffic sample in Example 1.

Example 1. Given a GET request with the URL as: <http://www.test.com/scripts/index.php?id=1' union select 1,2,3 from admin#>, the obtained HTTP traffic sample is $s = (x, y)$, where x is the path of the URL (i.e., [/scripts/index.php?id=1' union select 1,2,3 from admin#](#)) and y is SQL injection (i.e., a type of malicious HTTP traffic).

Second, we define the problem in this paper as follows. Given a large public labeled dataset D^{PU} and a small dataset from the target system D^{TS} (including both labeled and unlabeled samples), the malicious HTTP traffic detection problem can be considered as learning a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ using D^{PU} and D^{TS} . For any unseen sample from the target system $s^{\text{TS}} = (x^{\text{TS}}, y^{\text{TS}})$, $f(x^{\text{TS}})$ and y^{TS} should be as similar as possible. Here, \mathcal{X} represents the input space and \mathcal{Y} represents the set of labels.

Third, Figure 1 gives the architecture of DeepPTSD, which consists of three major steps, i.e., model initialization, data augmentation, and transfer semi-supervised learning. First, the model initialization step trains the initial model, with DPU as training data and parameters being randomly initialized. Here, we treat malicious HTTP traffic detection as a special text classification problem and utilize TextCNN as the underlying deep neural network (detailed in Section 3.2). Second, the data augmentation step enhances the dataset of the target system by generating a diverse set of samples from D^{TS} . Here, we propose a HTTP payload data augmentation method based on a keyword avoidance perturbation mechanism for this task (detailed in Section 3.3). Third, the transfer semi-supervised learning step adapts the initial model to the target system by fine-tuning it on the augmented dataset of D^{TS} . In this paper, we use the same deep neural network (i.e., TextCNN) for both the initial model and the model of the target system (i.e., target model). Here, we design a learning framework by integrating a semi-supervised learning framework (i.e., the UDA framework detailed in Section 3.4) into a transfer learning structure, to leverage the augmented samples.

In practice, the malicious HTTP payload detection model can be deployed as follows. A user consults a web application and sends a request to the webserver. Before the request arrives at the server, it will be processed by a WAF (Web Application Firewall), which takes as input all the HTTP requests and processes them using the proposed malicious HTTP payload detection model. If the request is malicious then it will be rejected automatically, else it will be sent to the webserver.

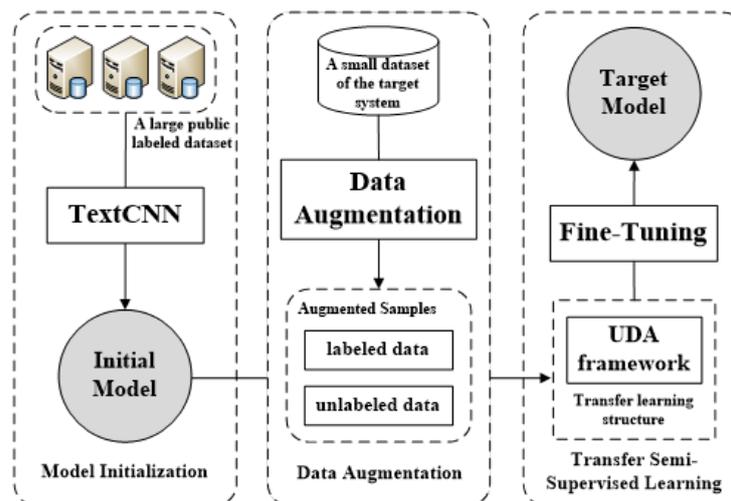


Figure 1. The architecture of DeepPTSD.

3.2. Model Initialization

The purpose of the model initialization step is to train the initial malicious HTTP traffic detection model based on the public labeled dataset D^{PU} . Since we focus on the payload of single HTTP requests, and payload analysis task is similar with NLP (Natural Language Processing) task [32], we treat malicious HTTP traffic detection as a special text classification problem.

First, given a HTTP traffic sample $s = (x, y)$, we have to segment x into discrete tokens before inputting it into a text classifier. Unlike English sentences that can be naturally segmented into words by space and punctuation, the payload of a HTTP traffic sample is an integrated string. Although some studies used single characters [12,13] or N-gram algorithm [8,19] to segment an integrated string, it would result in a large number of tokens without semantic meanings. For instance, if we set $N = 3$, the N-gram algorithm would segment the payload in Example 1 into many meaningless tokens (e.g., htt, est, cri), which make it difficult for the text classifiers to learn real semantic patterns. Fortunately, the payload of a HTTP traffic sample usually involves a variety of special characters (e.g., ?, :, /). Therefore,

we use the 26 special characters in HTTP traffic (i.e., ', ", <, >, +, -, _ *, =, {, }, (,), [,], ~, /, \, #, :, ;, ?, !, -, and, @) as the basis for segmentation. We give an example of payload segmentation in Example 2.

Example 2. The payload in Example 1 is segmented into [http](#) [www](#) [test](#) [com](#) [scripts](#) [index](#) [php](#) [id](#) [1](#) [union](#) [select](#) [1](#) [2](#) [3](#) [from](#) [admin](#). It can be seen that tokens that have strong semantics related to malicious behaviors (e.g., *select*, *union*) are correctly segmented.

Second, we apply TextCNN proposed in [33] to train the initial detection model, since it shows a good balance between classification accuracy and training efficiency for the text classification tasks. As shown in Figure 2, TextCNN is composed of four sequentially stacked layers, i.e., an embedding layer, a convolutional layer, a pooling layer, and an output layer. (1) The embedding layer is used to transform each discrete token into a low-dimensional semantic vector, so that tokens with similar semantics would be close in vector space. Here, we utilize word2vec [34] as the embedding technique. Given a HTTP traffic sample $s = (x, y)$, let the length of x be l and the dimension of the semantic vectors be d , x can then be represented as an $l \times d$ matrix. Here, we denote the semantic vector corresponding to the i th token in x as v_i . Note that the payload would be padded or truncated if it was shorter or longer than l . (2) The convolutional layer uses multiple filters with different window sizes to extract feature maps. Specifically, a filter with window size $h \times d$ is applied to each possible contiguous h tokens in x (i.e., the concatenation of the semantic vectors $v_i, v_{i+1}, \dots, v_{i+h-1}$, denoted as $v_{i:i+h-1}$) to

produce a feature value $c_i = \text{ReLU}(W \cdot v_{i:i+h-1} + b)$, where W is the weight matrix and b is the bias term. Thus, each filter would generate a feature vector $c = [c_1, c_2, \dots, c_{l-h+1}]$. (3) The pooling layer applies a max-over-time pooling operation over each feature vector, i.e., it takes the maximum value of the feature vector as the feature value corresponding to the particular filter. Then, by concatenating the pooling results of all filters, we can obtain a final feature vector with uniform length for different payloads (i.e., the length equals to the number of filters). (4) The output layer accepts the final feature vector and output the probability distribution over all the HTTP traffic types based on a fully connected operation.

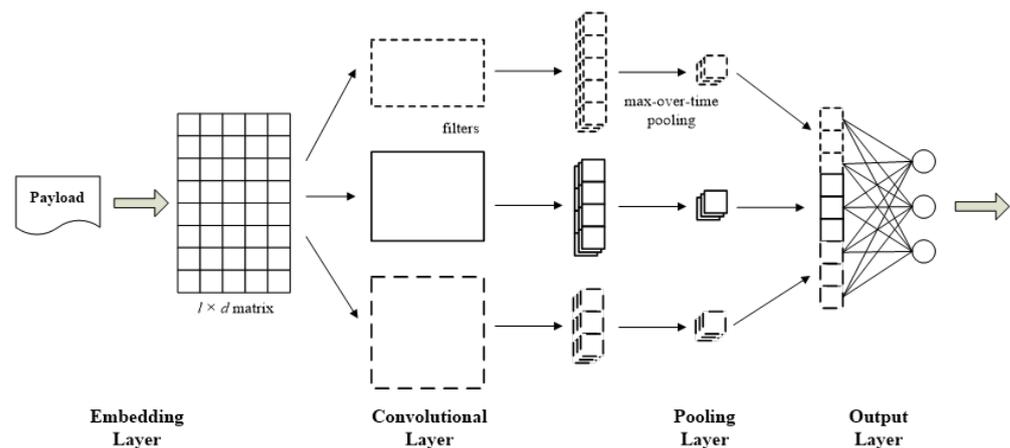


Figure 2. The architecture of TextCNN.

3.3. Data Augmentation

Since the dataset of the target system DTS is usually limited in size and of low degree of diversity, retraining or fine-tuning the initial detection model on such dataset would easily lead to overfitting. Data augmentation is a process of generating more data with higher degree of diversity by using the available limited data, so as to address the overfitting issue [35].

Since payloads are in the form of textual data, it is intuitive to apply textual data augmentation methods for the payload data augmentation task. The existing textual data augmentation methods include back translation [36,37], word replacement [38,39], and sentence generation [35,40]. Back translation is a procedure of translating into another language and then back to the original language. It can generate diverse paraphrases while preserving the semantics of the original sentence. However, payloads are not human language, and thus cannot be translated. Word replacement generates new sentences by replacing selected words in the original sentence with new words, and the most intuitive strategy is replacing with synonym. However, a large portion of tokens in payloads have particular meaning. For example, if we replace “select” in the payload of Example 1 with “choose”, the new payload `/scripts/index.php?id=1'union choose 1,2,3 from admin#` would lose the ability of web attack, because “select” is a particular keyword for querying the database. Sentence generation uses generative model such as VAE (variational autoencoder) and GAN (generative adversarial network) to generate sentences from a latent space. However, it is difficult to learn a conditional token distribution from the payloads, since they do not have strong structure and grammar patterns as compared with the natural human language.

Based on the above analysis, the existing textual data augmentation methods cannot be directly used for payload data augmentation task. Aiming at this problem, we propose a HTTP payload data augmentation method based on a keyword avoidance perturbation mechanism. The idea is to add noises to the original payloads while keeping the particular keywords unchanged. The proposed HTTP payload data augmentation method consists of two steps, i.e., keyword mining and payload perturbation.

Keyword mining: A simple approach is to manually build a keyword library for HTTP payloads [12,14]. However, this approach has the following limitations. First, manual building is tedious and costly, and it usually has a very low recall rate. Second, attackers could use obfuscation operation to change the keywords in a payload into meaningless substrings, in order to bypass the malicious keyword detection system. As shown in Example 3, it is extremely difficult to manually collect these obfuscated keywords.

Example 3. The keywords “select” and “union” could be changed into “/*!50000%53elect*/” and “uni%0bon”, respectively. Then, the payload in Example 1 can be transformed into: http://www.test.com/scripts/index.php?id=1' uni%0bon /*!50000%53elect*/ 1,2,3 from admin#.

Therefore, we use a TF-IDF based method for keyword mining. First, given a malicious HTTP payload dataset MD and a benign HTTP payload dataset BD , where each payload has been segmented based on the method in Section 3.2, we calculate the TF-IDF score $tis(w)$ for each token w based on Equation (1), where $n_{MD}(w)$ is the number of w in MD , n_{MD} is the number of all tokens in MD , and d_j is the j th payload in BD . According to Equation (1), a token that frequently appears in malicious payloads but seldom or never appears in benign payloads would be given a high TF-IDF score. Second, for each token w , if its TF-IDF score $tis(w)$ is larger than a predefined threshold σ , it will be defined as a keyword. Here, we give a partial of the mined keywords as follows: “script”, “cgi”, “bin”, “path”, “cookie”, “cross”, “meta”, “select”, “where”, “from”, etc.

$$tis(w) = \frac{n_{MD}(w)}{n_{MD}} \times \log \frac{|BD|}{|\{j : w \in d_j\}| + 1} \quad (1)$$

Payload perturbation: Once the keyword library is constructed, we generate new payloads from the original payloads by retaining keywords and adding noises to the uninformative tokens based on the rules in Table 1. Note that we adopt character-level randomization rather than word-level replacement in Rule 4, because most tokens in the payloads do not have natural language semantics (e.g., www, index, admin in the payload of Example 1). We give an example of payload perturbation in Example 4.

Table 1. The rules of payload perturbation.

Rule 1	Keep all keywords in the keyword library unchanged.
Rule 2	Retain all the special characters (e.g., %, @, #).
Rule 3	Randomize each number in the range of [0,9].
Rule 4	Change each letter into a random letter.

Example 4. Suppose that the keyword library contained four keywords “union”, “select”, “from”, and “scripts”, the payload in Example 1 could generate a new payload after perturbation as: [/scripts/raonb.bvp?br=7' union select 4,7,2 from loogs#](#).

The dataset of the target system D^{TS} includes a labeled dataset D^{TSL} and an unlabeled dataset D^{TSU} . In practice, D^{TSL} is usually of a very small size and D^{TSU} is usually of a relatively larger size. We perform payload data augmentation on both D^{TSL} and D^{TSU} . We denote the generated labeled dataset and the generated unlabeled dataset as GD^{TSL} and GD^{TSU} , respectively.

3.4. Transfer Semi-Supervised Learning

We transfer the initial detection model to the target system by fine-tuning it on $D^{TSL} \cup D^{TSU} \cup GD^{TSL} \cup GD^{TSU}$. Here, fine-tuning is known as a process that copies the parameters of the initial model to initialize the corresponding part of the target model and then the target model is retrained through backpropagation with target training samples [28]. The fine-tuning process is performed based on the proposed transfer semi-supervised learn-

ing framework, which is designed by integrating a semi-supervised learning framework into a transfer learning structure. We explain the details as follows.

First, the transfer learning structure is designed based on the idea proposed in [28]. The parameters of the initial model are copied to the target model. Then, the first n layers of the target model are frozen, and then the parameters of the rest layers are retrained based on the dataset of the target system. A frozen layer means that it does not change during retraining. We explain the idea as follows. (1) We do not retrain all the layers, since this would cause catastrophic forgetting, eliminating the knowledge of the initial model. Especially, if the dataset of the target system is small, retraining all layers would easily result in overfitting [41]. (2) We leave the last a few layers unfrozen, since they contain the most specific knowledge that should be learnt from the target system [28]. For example, if the public dataset and the dataset of target system have different types of malicious HTTP traffic, the last layer of TextCNN (i.e., the output layer) is mandatory to be retrained.

Second, to better exploit the labeled and unlabeled samples from the target system, we conduct the fine-tuning process in a semi-supervised learning style. In all the deep semi-supervised learning styles, the smoothness enforcing method tries to regularize the model's prediction to be less sensitive to small perturbations [41–43]. The smoothness enforcing method is the most suitable deep semi-supervised learning style for our problem, since it is able to exploit all the labeled, unlabeled, and augmented samples. Specifically, given an original sample s_o , the smoothness enforcing method creates a perturbed version of s_o (denoted as s_p), and then trains the model to output similar predictions on s_o and s_p . In our problem, the payload data augmentation operation corresponds to the perturbation operation of the smoothness enforcing method. To this end, we leverage and refine the UDA framework proposed in [43] to perform the smoothness enforcing based semi-supervised learning. Specifically, we fine-tune the initial model to minimize the loss function in Equation (2), where $LL(D^{\text{TSL}} \cup GD^{\text{TSL}})$ and $UL(D^{\text{TSU}} \cup GD^{\text{TSU}})$ denote the loss on labeled datasets (i.e., D^{TSL} and GD^{TSL}) and unlabeled datasets (i.e., D^{TSU} and GD^{TSU}), λ is a trade-off weight, $p_\theta(x)$ denotes the predicted label distribution of sample x based on the current model parameters θ , $\delta(x)$ is the real label distribution of sample x (i.e., a one-shot vector where only the entry corresponding to the real label is 1), $x^* \sim A_{\text{PL}}(x)$ represents the augmented sample generated from sample x , and $D_{\text{KL}}(p_\theta(x) \| p_\theta(x^*))$ represents KL divergence between $p_\theta(x)$ and $p_\theta(x^*)$.

$$L = LL(D^{\text{TSL}} \cup GD^{\text{TSL}}) + \lambda \times UL(D^{\text{TSU}} \cup GD^{\text{TSU}}) \quad (2)$$

$$LL(D^{\text{TSL}} \cup GD^{\text{TSL}}) = - \sum_{x \in D^{\text{TSL}} \cup GD^{\text{TSL}}} \delta(x) \cdot \log p_\theta(x) \quad (3)$$

$$UL(D^{\text{TSU}} \cup GD^{\text{TSU}}) = \sum_{x \in D^{\text{TSU}}, x^* \sim A_{\text{PL}}(x) \in GD^{\text{TSU}}} D_{\text{KL}}(p_\theta(x) \| p_\theta(x^*)) \quad (4)$$

Finally, Figure 3 shows the structure of the proposed transfer semi-supervised learning framework, and we summarize DeepPTSD in Algorithm 1.

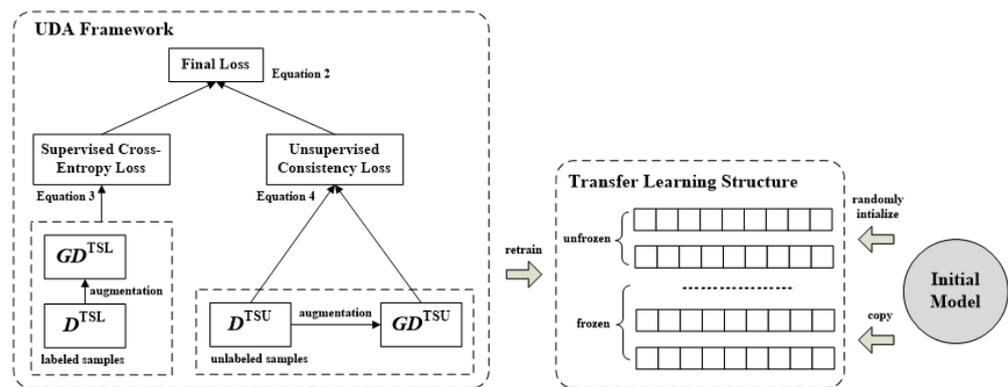


Figure 3. The proposed transfer semi-supervised learning framework.

Algorithm 1 DeepPTSD

Input: The public labeled dataset, D^{PU}
 The labeled dataset of the target system, D^{TSL}
 The unlabeled dataset of the target system, D^{TSU}
 The number of frozen layers, n
 The trade-off weight, λ

Output: The detection model of the target system, TM

- 1: Train an initial TextCNN classifier (denoted as IM) with D^{PU}
- 2: Generate an augmented dataset from D^{TSL} (denoted as GD^{TSL})
- 3: Generate an augmented dataset from D^{TSU} (denoted as GD^{TSU})
- 4: Initialize the TextCNN classifier of the target system (denoted as TM), by copying the parameters of IM to TM
- 5: Freeze the first n layers of TM and retrain TM based on D^{TSL} , D^{TSU} , GD^{TSL} , and GD^{TSU} by following Equation (2)

4. Experiment

4.1. Experiment Setup

4.1.1. Dataset

To conduct the experiments, we collected two raw datasets. One was an open dataset from FSecurify [44], which contained a large number of HTTP requests to a WAF (web application firewall). Only two types of HTTP requests were labeled (i.e., benign and malicious), and thus it did not discriminate among different types of malicious HTTP requests. The other one was collected from a honeypot server deployed by our lab during March, 2019. It contained 6465 HTTP traffic samples, including 879 benign samples, 614 SQL injection samples, 4093 XSS samples, and 879 directory traversal samples. To balance the data of different types, we resampled the above two raw datasets and form the following two datasets.

(1) Dataset A: It contained 80,000 samples (including 40,000 benign samples and 40,000 malicious samples), which were randomly selected from the open dataset.

(2) Dataset B: It contained 800 samples (including 200 benign samples, 200 SQL injection samples, 200 XSS samples, and 200 directory traversal samples), which were randomly selected from the honeypot dataset.

4.1.2. Evaluation Strategy

First, the malicious HTTP traffic detection is essentially a classification problem. Thus, we measured the performance based on three metrics, i.e., Precision, Recall, and F1-Measure.

Second, this paper focuses on cross-system malicious HTTP traffic detection problem. Therefore, we treated Dataset A as the large public labeled dataset and Dataset B as the small dataset of the target system. In addition, Dataset B was further divided into two subsets, i.e., the labeled subset (called Subset BL) and the unlabeled subset (called Subset

BU). Subset BL contained 200 samples (i.e., 50 samples for each type) and Subset BU contained 600 samples (i.e., 150 samples for each type). To cope with different evaluation scenarios, we designed the following two evaluation strategies.

(1) Strategy A: It corresponded to the standard training and testing strategy. Specifically, it trained the target model on Subset BL and tested it on Subset BU.

(2) Strategy B: It corresponded to the training and testing strategy for the small data condition. Specifically, it trained the target model on Subset BL \cup Subset BU (corresponding to the semi-supervised learning scenario), Subset BL \cup Dataset A (corresponding to the transfer learning scenario), or Subset BL \cup Subset BU \cup Dataset A (corresponding to the transfer semi-supervised learning scenario). Then, it tested the target model on Subset BU.

For both strategy A and strategy B, we adopted a four-fold-cross-validation procedure, i.e., we randomly generated Subset BL and Subset BU four times and reported the average performance.

4.1.3. Parameter Settings

Through a series of grid search experiment, we set the parameters of the proposed method as follows. For the TextCNN in Section 3.2, we set the length of the input payload $l = 64$ and the dimension of the word embeddings $d = 32$. We defined three window sizes (i.e., $h \times d$, $h = 2, 3$, and 4) for the convolutional filters. For the keyword mining algorithm in Section 3.3, we set the TF-IDF score threshold $\sigma = 0.5$ based on a grid search experiment. For the transfer semi-supervised learning framework in Section 3.4, we set the number of frozen layers $n = 2$ and the trade-off weight of the loss function $\lambda = 1$ based on the previous experiences.

4.2. Experiment 1: The Evaluation of Different Payload Segmentation Strategies

In this experiment, we tried to evaluate the effectiveness of the proposed payload segmentation approach (abbreviated as **SegAsWord**) by comparing it with the following two payload segmentation approaches, which were utilized in previous work.

- (1) **SegAsChar**: It corresponded to the character-level payload segmentation approach [21].
- (2) **SegAsNGram**: It corresponded to the approach that segments payloads based on the character-level N-gram algorithm [9]. Here, we set $N = 3$.

Here, strategy A was used for evaluation and TextCNN is used as the classifier. The experiment results are shown in Table 2. First, SegAsNGram outperforms SegAsChar. It was intuitive that N-gram fragments could better capture the payload semantics than single characters did. For example, if there was a 3-gram fragment “sel”, it was probably a SQL injection payload with the keyword “select”, while it was almost impossible to draw a reasonable conclusion if only a single character “s” was observed. However, although single characters could not capture the payload semantics, the performance of SegAsChar was not that bad. It is because that the convolution operation of TextCNN processed the adjacent h characters in a filter, which could achieve a similar effect with character-level N-gram. Second, it was also intuitive that SegAsWord outperformed SegAsNGram, because the segmented keywords could capture the payload semantics in an explicit manner.

Table 2. The evaluation of the payload segmentation approaches.

	SegAsChar	SegAsNGram	SegAsWord
Precision	0.7522	0.8134	0.8667
Recall	0.7893	0.8253	0.8722
F1-Measure	0.7703	0.8193	0.8694

4.3. Experiment 2: Ablation Experiment

In the first experiment, we tried to evaluate the data augmentation module of DeepPTSD. We compared the proposed payload data augmentation mechanism (abbreviated as **AugByKeyword**) with the following three variants of payload data augmentation mechanisms.

- (1) **NoAug**: It corresponded to the variant that no payload data augmentation was performed.
- (2) **RandomAug**: It corresponded to the variant that the noises were added to all the tokens without retraining the keywords. Note that the special characters (e.g., %, @, #) were retained.
- (3) **AugByDic**: It corresponded to the variant that the keywords needed to be retrained were predefined in a dictionary instead of being extracted from the dataset.

Here, strategy A was used for evaluation and TextCNN was used as the classifier. The experiment results are shown in Table 3. First, RandomAug had the worst performance and it performed even worse than NoAug. It showed that adding randomized noises without considering the keywords would not increase the diversity of the training samples but even damaged the semantic integrity of the payloads. It verified the effectiveness of the keyword avoidance perturbation mechanism. Second, AugByDic and AugByKeyword outperformed NoAug. It demonstrated the necessity of the payload data augmentation module. Third, AugByKeyword had only a slight advantage over AugByDic. However, AugByDic required manual efforts for defining the dictionary, and thus AugByKeyword had stronger scalability.

Table 3. The evaluation of the payload data augmentation module.

	NoAug	RandomAug	AugByDic	AugByKeyword
Precision	0.8667	0.8020	0.9012	0.9107
Recall	0.8722	0.7944	0.8847	0.8913
F1-Measure	0.8694	0.7982	0.8929	0.9009

In the second experiment, we tried to evaluate the transfer learning module and the semi-supervised learning module. Given the five datasets D^{PU} , D^{TSL} , D^{TSU} , GD^{TSL} , and GD^{TSU} (specified in Algorithm 1), we compared the performance of the following three variants of DeepPTSD.

- (1) **DeepPTSD_None**: It referred to the standard deep learning model. Specifically, it trained the target detection model on D^{TSL} based on TextCNN (in Section 3.2).
- (2) **DeepPTSD_T**: It referred to the transfer learning model with augmented labeled data. Specifically, it trained the initial detection model on D^{PU} , and fine-tuned it on $D^{TSL} \cup GD^{TSL}$ to get the target detection model based on the transfer learning structure (in Section 3.4). Here, the first two layers of the initial detection model were frozen and we fine-tuned it to minimize the loss function in Equation (3).
- (3) **DeepPTSD_S**: It referred to the semi-supervised learning model with augmented labeled and unlabeled data. Specifically, it trained the target detection model on $D^{TSL} \cup GD^{TSL} \cup D^{TSU} \cup GD^{TSU}$ based on the UDA framework (in Section 3.4).

Here, strategy A was used for evaluating DeepPTSD_None, while strategy B was used for evaluating DeepPTSD, DeepPTSD_T, and DeepPTSD_S. All the methods were tested on D^{TSU} . The experiment results are shown in Table 4 to control sequence. First, DeepPTSD_T outperformed DeepPTSD_None and DeepPTSD_S outperformed AugByKeyword and DeepPTSD_None. It demonstrated that both the transfer learning module and the semi-supervised learning module contributed to the detection model. As a result, DeepPTSD had the best performance. Second, AugByKeyword and DeepPTSD_A outperformed DeepPTSD_T. It indicated that the data augmentation and semi-supervised learning modules were more effective for the detection model than

the transfer learning module. This result might be affected by the experiment dataset. Transfer learning usually requires a very large source domain dataset to gain its power (e.g., the pre-training language model such as BERT and GPT). When the source domain dataset is limited, leveraging the unlabeled data and performing data augmentation directly on target domain dataset could achieve better results.

Table 4. The evaluation of the payload data augmentation module.

	DeepPTSD_None	DeepPTSD_T	DeepPTSD_S	DeepPTSD
Precision	0.8667	0.9067	0.9223	0.9333
Recall	0.8722	0.8864	0.9207	0.9348
F1-Measure	0.8694	0.8964	0.9215	0.9340

4.4. Experiment 3: Comparison Experiment

To evaluate the competitive performance of DeepPTSD, we compared it with the following five baselines.

- (1) **SVM_C**: It corresponded to the character-level feature based method. Specifically, it first segmented each HTTP traffic sample into characters, and built a feature vector to reflect the character distribution based on VSM (vector space model). Then, it trained a SVM (support vector machine) classifier based on the character feature vectors.
- (2) **SVM_W**: It corresponded to the word-level feature based method [18]. Specifically, it first segmented each HTTP traffic sample into words based on the approach proposed in Section 3.2, and built a feature vector to reflect the word distribution based on VSM. Then, it trained a SVM classifier based on the word feature vectors.
- (3) **CNN**: It trained the detection model based on the TextCNN model [14]. Here, a HTTP traffic sample was segmented based on the approach proposed in Section 3.2.
- (4) **RNN**: It trained the detection model based on a BiLSTM model [22]. Here, a HTTP traffic sample was also segmented based on the approach proposed in Section 3.2.
- (5) **AE**: It corresponded to the autoencoder based method [12]. Specifically, it first trained an autoencoder on $D^{PU} \cup D^{TSL} \cup D^{TSU}$ in an unsupervised manner to map each HTTP traffic sample into a unified latent feature space. Then, it trained a MLP (multi-layer perceptron) classifier based on the latent feature vectors.

Here, strategy A was used for evaluating SVM_C, SVM_W, CNN, and RNN, while strategy B was used for evaluating AE and DeepPTSD. AE was trained on D^{PU} , D^{TSL} , and D^{TSU} , while DeepPTSD is trained on D^{PU} , D^{TSL} , D^{TSU} , GD^{TSL} , and GD^{TSU} . AE can be viewed as an alternative way to implement the transfer semi-supervised learning. The experiment results are shown in Table 5 and the following tendencies can be discerned. First, SVM_C had a far lower performance than SegAsChar and SVM_W. It indicated that single characters could not capture the payload semantics. Although SegAsChar also accepted single characters as input, the convolution operation of TextCNN could process multiple adjacent characters in a filter. Second, CNN and RNN outperformed SVM_W. It showed that deep learning based methods could extract more advanced features to capture the payload semantics. Third, AE outperformed CNN and RNN. It verified the effectiveness of the utilization of source domain data and unlabeled target domain data, which could provide richer knowledge to improve the generalization ability of the detection model when the labeled target domain data are insufficient. Finally, DeepPTSD had the best performance. It showed that it could achieve a better performance by integrating transfer learning, data augmentation, and semi-supervised learning.

Table 5. The evaluation of the payload data augmentation module.

	SVM_C	SVM_W	CNN	RNN	AE	DeepPTSD
Precision	0.6103	0.8139	0.8667	0.8661	0.8936	0.9333
Recall	0.6324	0.7717	0.8722	0.8746	0.8816	0.9348
F1-Measure	0.6212	0.7922	0.8694	0.8703	0.8876	0.9340

5. Conclusions

In this paper, aiming at the huge training data collection cost and low cross-system detection ability of the existing malicious HTTP traffic detection methods, we propose DeepPTSD, a payload based malicious HTTP traffic detection method based on deep transfer semi-supervised learning technique. It improves the generalization ability of the detection model under the situation of insufficient training data by using two strategies. The first is a transfer learning strategy, which learns generalized knowledge from a public dataset and adapts the knowledge to the target system. The second is a semi-supervised learning strategy, which performs data augmentation by using a keyword avoidance perturbation mechanism and trains the detection model in a smoothness enforcing style by exploiting all the labeled, unlabeled, and augmented target domain data. Through a series of experiments based on real datasets, we demonstrate that DeepPTSD outperforms the existing baselines when the training dataset is highly limited.

In the future, we will extend our work from the following directions. First, the dataset used in the experiments only contains simple types of web attacks (e.g., SQL injection, XSS). We will extend our method to detect advanced web attacks (e.g., webshell). Second, we will fuse our method with statistic based method to detect encrypted malicious HTTP traffic. Third, the malicious HTTP traffic detection is treated as a text classification task, and only the payload data in the form of string are considered in the detection model. In the future, we will extend our method to accommodate other elements of the HTTP protocol (e.g., request header, response header, etc.). Fourth, we will study the anomaly detection based method for the malicious HTTP traffic detection problem, in case no malicious samples are available.

Author Contributions: Methodology, M.L.; Software, Y.C. and G.H.; Supervision, T.C.; Writing—review and editing, T.Z., T.W. and Z.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Joint Funds of the National Natural Science Foundation of China (No. U1936215), Science Foundation of China (No. 62002324), Zhejiang Provincial Natural Science Foundation of China (No. LQ21F020016), Primary R&D Plan of Zhejiang Province (No. 2021C01117), the Innovative Development of the Industrial Internet Project (No. TC200H01V), and Innovative Leading Talent Project of Zhejiang Province (No. 2020R52001).

Data Availability Statement: The data can be found at <https://github.com/faizann24/Using-machine-learning-to-detect-malicious-URLs>, accessed on 28 July 2021.

Acknowledgments: This work was supported by the Joint Funds of the National Natural Science Foundation of China (No. U1936215), the Natural Science Foundation of China (No. 62002324), the Zhejiang Provincial Natural Science Foundation of China (Nos. LQ21F020016, LY20F020027), the Primary Research and Development Plan of Zhejiang Province (No. 2021C01117), the Innovative Development of the Industrial Internet Project (No. TC200H01V), and the Wenzhou Key Scientific and Technological Project (No. ZG2020031).

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

References

1. Rotsos, C.; Van Gael, J.; Moore, A.W.; Ghahramani, Z. Probabilistic graphical models for semi-supervised traffic classification. In Proceedings of the 6th International Wireless Communications and Mobile Computing Conference, Caen, France, 28 June–2 July 2010; pp. 752–757.
2. Jadidi, Z.; Muthukkumarasamy, V.; Sithirasenan, E.; Singh, K. Flow-based anomaly detection using semisupervised learning. In Proceedings of the 2015 9th International Conference on Signal Processing and Communication Systems (ICSPCS), Cairns, QLD, Australia, 14–16 December 2015; pp. 1–5.
3. Draper-Gil, G.; Lashkari, A.H.; Mamun, M.S.I.; Ghorbani, A.A. Characterization of encrypted and vpn traffic using time-related. In Proceedings of the 2nd International Conference on Information Systems Security and Privacy (ICISSP), Rome, Italy, 19–21 February 2016; pp. 407–414.
4. Bapat, R.; Mandya, A.; Liu, X.; Abraham, B.; Brown, D.E.; Kang, H.; Veeraraghavan, M. Identifying malicious botnet traffic using logistic regression. In Proceedings of the 2018 Systems and Information Engineering Design Symposium (SIEDS), Charlottesville, VA, USA, 27 April 2018; pp. 266–271.
5. Shekhawat, A.S.; Di Troia, F.; Stamp, M. Feature analysis of encrypted malicious traffic. *Expert Syst. Appl.* **2019**, *125*, 130–141. [[CrossRef](#)]
6. Zhenqi, W.; Xinyu, W. Netflow based intrusion detection system. In Proceedings of the 2008 International Conference on Multimedia and Information Technology, Yichang, China, 30–31 December 2008; pp. 825–828.
7. Nguyen, T.T.; Armitage, G. A survey of techniques for internet traffic classification using machine learning. *IEEE Commun. Surv. Tutor.* **2008**, *10*, 56–76. [[CrossRef](#)]
8. Machlica, L.; Bartos, K.; Sofka, M. Learning detectors of malicious web requests for intrusion detection in network traffic. *arXiv* **2017**, arXiv:1702.02530.
9. Verma, R.; Das, A. What's in a url: Fast feature extraction and malicious url detection. In Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics, Scottsdale, AZ, USA, 24 March 2017; pp. 55–63.
10. Liu, C.; Wang, L.; Lang, B.; Zhou, Y. Finding effective classifier for malicious URL detection. In Proceedings of the 2018 2nd International Conference on Management Engineering, Software Engineering and Service Sciences, Wuhan, China, 13–15 January 2018; pp. 240–244.
11. Patgiri, R.; Katari, H.; Kumar, R.; Sharma, D. Empirical study on malicious URL detection using machine learning. In *Proceedings of the International Conference on Distributed Computing and Internet Technology*; Springer: Cham, Switzerland, 2019; pp. 380–388.
12. Park, S.; Kim, M.; Lee, S. Anomaly detection for HTTP using convolutional autoencoders. *IEEE Access* **2018**, *6*, 70884–70901. [[CrossRef](#)]
13. Peng, Y.; Tian, S.; Yu, L.; Lv, Y.; Wang, R. A joint approach to detect malicious URL based on attention mechanism. *Int. J. Comput. Intell. Appl.* **2019**, *18*, 1950021. [[CrossRef](#)]
14. Le, H.; Pham, Q.; Sahoo, D.; Hoi, S.C. URLNet: Learning a URL representation with deep learning for malicious URL detection. *arXiv* **2018**, arXiv:1802.03162.
15. Yang, W.; Zuo, W.; Cui, B. Detecting malicious urls via a keyword-based convolutional gated-recurrent-unit neural network. *IEEE Access* **2019**, *7*, 29891–29900. [[CrossRef](#)]
16. Liu, H.; Lang, B. Machine learning and deep learning methods for intrusion detection systems: A survey. *Appl. Sci.* **2019**, *9*, 4396. [[CrossRef](#)]
17. Wang, C.; Guan, X.; Qin, T.; Wang, P.; Tao, J.; Meng, Y.; Liu, J. Addressing the train–test gap on traffic classification combined subflow model with ensemble learning. *Knowl. Based Syst.* **2020**, *204*, 106192. [[CrossRef](#)]
18. Ma, J.; Saul, L.K.; Savage, S.; Voelker, G.M. 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, Paris, France, 28 June–1 July 2009; pp. 1245–1254.
19. Yang, J.; Wang, L.; Xu, Z. A novel semantic-aware approach for detecting malicious web traffic. In *Proceedings of the International Conference on Information and Communications Security*; Springer: Cham, Switzerland, 2017; pp. 633–645.
20. Yang, J.; Yang, P.; Jin, X.; Ma, Q. Multi-classification for malicious URL based on improved semi-supervised algorithm. In Proceedings of the 2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), Guangzhou, China, 21–24 July 2017; Volume 1, pp. 143–150.
21. Saxe, J.; Berlin, K. eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *arXiv* **2017**, arXiv:1702.08568.
22. Yu, Y.; Liu, G.; Yan, H.; Li, H.; Guan, H. Attention-based Bi-LSTM model for anomalous HTTP traffic detection. In Proceedings of the 2018 15th International Conference on Service Systems and Service Management (ICSSSM), Hangzhou, China, 21–22 July 2018; pp. 1–6.
23. Li, Z.; Yao, H.; Ma, F. Learning with Small Data. In Proceedings of the 13th International Conference on Web Search and Data Mining, Houston, TX, USA, 3–7 February 2020; pp. 884–887.
24. Long, Y.; Liu, L.; Shen, F.; Shao, L.; Li, X. Zero-shot learning using synthesised unseen visual data with diffusion regularisation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *40*, 2498–2512. [[CrossRef](#)]
25. Antoniou, A.; Storkey, A.; Edwards, H. Data augmentation generative adversarial networks. *arXiv* **2017**, arXiv:1711.04340.

26. Chen, Z.; Fu, Y.; Zhang, Y.; Jiang, Y.G.; Xue, X.; Sigal, L. Semantic feature augmentation in few-shot learning. *arXiv* **2018**, arXiv:1804.05298.
27. Ramachandram, D.; Taylor, G.W. Deep multimodal learning: A survey on recent advances and trends. *IEEE Signal Process. Mag.* **2017**, *34*, 96–108. [[CrossRef](#)]
28. Yosinski, J.; Clune, J.; Bengio, Y.; Lipson, H. How transferable are features in deep neural networks? *arXiv* **2014**, arXiv:1411.1792.
29. Hinton, G.; Vinyals, O.; Dean, J. Distilling the knowledge in a neural network. *arXiv* **2015**, arXiv:1503.02531.
30. Davis, E.; Marcus, G. Commonsense reasoning and commonsense knowledge in artificial intelligence. *Commun. ACM* **2015**, *58*, 92–103. [[CrossRef](#)]
31. Stewart, R.; Ermon, S. Label-free supervision of neural networks with physics and domain knowledge. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
32. Fang, Y.; Xu, Y.; Jia, P.; Huang, C. Providing Email Privacy by Preventing Webmail from Loading Malicious XSS Payloads. *Appl. Sci.* **2020**, *10*, 4425. [[CrossRef](#)]
33. Kim, Y. ConvolutionalNeuralNetworksforSentence Classification. *arXiv* **2014**, arXiv:1408.5882.
34. Google. Word2vec. 2013. Available online: <https://code.google.com/p/word2vec/> (accessed on 28 February 2021)
35. Qiu, S.; Xu, B.; Zhang, J.; Wang, Y.; Shen, X.; de Melo, G.; Long, C.; Li, X. EasyAug: An automatic textual data augmentation platform for classification tasks. In *Companion Proceedings of the Web Conference 2020*; ACM: New York, NY, USA, 2020; pp. 249–252.
36. Silfverberg, M.; Wiemerslage, A.; Liu, L.; Mao, L.J. Data augmentation for morphological reinflection. In Proceedings of the CoNLL SIGMORPHON 2017 Shared Task: Universal Morphological Reinflection, Vancouver, BC, Canada, 3–4 August 2017; pp. 90–99.
37. Yu, A.W.; Dohan, D.; Luong, M.T.; Zhao, R.; Chen, K.; Norouzi, M.; Le, Q.V. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv* **2018**, arXiv:1804.09541.
38. Kobayashi, S. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv* **2018**, arXiv:1805.06201.
39. Wei, J.; Zou, K. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv* **2019**, arXiv:1901.11196.
40. Zhang, Y.; Gan, Z.; Carin, L. Generating text via adversarial training. In *NIPS Workshop on Adversarial Training*; Academia.edu: San Francisco, CA, USA, 2016; Volume 21, pp. 21–32.
41. Miyato, T.; Maeda, S.i.; Koyama, M.; Ishii, S. Virtual adversarial training: A regularization method for supervised and semi-supervised learning. *IEEE Trans. Pattern Anal. Mach. Intell.* **2018**, *41*, 1979–1993. [[CrossRef](#)] [[PubMed](#)]
42. Tarvainen, A.; Valpola, H. Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. *arXiv* **2017**, arXiv:1703.01780.
43. Xie, Q.; Dai, Z.; Hovy, E.; Luong, M.T.; Le, Q.V. Unsupervised data augmentation for consistency training. *arXiv* **2019**, arXiv:1904.12848.
44. FSecurify. Using-Machine-Learning-to-Detect-Malicious-URLs. 2017. Available online: <http://fsecurify.com/using-machine-learning-detect-malicious-urls/> (accessed on 31 January 2021).