information

MDPI

Article

DGA CapsNet: 1D Application of Capsule Networks to DGA Detection

Daniel S. Berman

Johns Hopkins University Applied Physics Laboratory (JHU/APL1), Laurel, MD 20723, USA; daniel.berman@jhuapl.edu

Received: 26 February 2019; Accepted: 23 April 2019; Published: 27 April 2019

Abstract: Domain generation algorithms (DGAs) represent a class of malware used to generate large numbers of new domain names to achieve command-and-control (C2) communication between the malware program and its C2 server to avoid detection by cybersecurity measures. Deep learning has proven successful in serving as a mechanism to implement real-time DGA detection, specifically through the use of recurrent neural networks (RNNs) and convolutional neural networks (CNNs). This paper compares several state-of-the-art deep-learning implementations of DGA detection found in the literature with two novel models: a deeper CNN model and a one-dimensional (1D) Capsule Networks (CapsNet) model. The comparison shows that the 1D CapsNet model performs as well as the best-performing model from the literature.

Keywords: deep learning; deep neural networks; capsule networks; convolutional neural networks; cybersecurity; domain generation algorithms

1. Introduction

Domain generation algorithms (DGAs) are a type of malware tool used by attackers to generate a large number of domain names on the fly. By generating a massive quantity of domain names as needed, attackers can hide their command and control (C2) server and evade detection by standard cyber security methods. This scheme, called domain fluxing, is similar to hiding the proverbial needle (the attacker's C2 server) in a haystack [a long list of Internet Protocol (IP) addresses] [1]. Prior to DGAs, malware used a static list of domain names, and cyber defenders neutralized the malware by blacklisting specific domain names. However, with the introduction of DGAs, the domains are constantly changing, and it becomes impossible for the cyber defender to block all of the attacker domains via a blacklist. Furthermore, reverse engineering a DGA is a time-consuming task even if the defender can achieve it. A more effective and faster approach involves the use of machine-learning techniques to identify and flag suspected malicious domains.

The initial attempts at developing DGA detectors were aimed at classifying the DGA using a variety of traits. McGrath and Gupta [2] used features such as "whois" records, lexical characteristics, and known malicious IP addresses. Other researchers employed time-based, domain name server (DNS) answer-based, and domain-based properties using J48 trees [3]. Features such as domain length and hostname, among others, were used to identify advertising spam [4]. Other researchers used features such as the distribution of characters, bigrams, and structural features of the domains, like length and word presence, and techniques like regression [5], Alternating Decision Trees [6], support vector machines (SVMs) [7], and evolving spiking neural networks (eSNNs) [8]. In addition, deep learning and sequences of universal resource locators (URLs) using domain-based features and momentary URL-based features have been used for DGA detection [9]. The main problem with these methods is that they typically require some prepro cessing and thus cannot be implemented effectively in real time.

The first real-time DGA classifier used a featureless Long Short-Term Memory (LSTM) network, a type of recurrent neural network (RNN) [10]. This approach requires no external features and treats each character of the domain as a feature. It has been used with a variety of deep-learning models including convolutional neural networks (CNNs)+LSTM [11], bidirectional LSTM [12] with embedding [11], simple one-dimensional (1D) CNNs with only a convolution layer and no maximum pooling layers [13], pre-trained CNN image classifiers [14], multiple CNNs in parallel [15], and class-imbalance LSTMs specifically for identifying classes of DGAs [16]. These are all models included in the review [17] that found they all performed similarly well. These achieved varying levels of success. In particular, the embedding, CNN, and RNN layers have successfully resolved a variety of language-related problems.

The focus of this study is to introduce the following two new real-time DGA detection models and to then replicate and compare the performance of the most successful of the supervised deep learning techniques presented in the literature to these models.

- A 1D version of Capsule Networks (CapsNet), a new CNN architecture that eliminates maximum pooling layers in favor of new capsule layers
- A 1D CNN that contains multiple layers, including maximum pooling layers

Although capsule networks were recently applied to text classification [18], to the author's knowledge, this is the first time that methods using capsule networks were applied to the language and text domain within the cybersecurity arena. Furthermore, because CNNs perform better with more layers, a model with a deeper 1D CNN architecture was developed.

The two models tested and featured in this paper are compared to four models from the literature, specifically LSTM [10], CNN+LSTM [11], bidirectional LSTM [11], and a shallow 1D CNN [13], and parallel CNNs [15]. These five models were chosen because they only used the raw domain name as input without feature engineering, as this facilitates real time detection of DGAs. Models that performed feature engineering were not considered. Additionally, the model using a pretrained CNN image classifier [14] was not selected for comparison because it was not as successful as the others. The bidirectional LSTM demonstrated in [12] was also not chosen because the addition of an embedding layer, as was implemented in [11], generally improves results.

The remainder of this paper is organized as follows: Section 2 provides background information on deep learning. Sections 3 and 4, respectively, describe the models that were tested in greater detail and the datasets that were used. Section 5 discusses the various metrics that were used to evaluate the models. Section 6 presents the models' training and testing results. Lastly, Section 7 presents the conclusions that can be drawn.

2. Background

This section provides a brief overview of the various concepts discussed in the introduction. A more in-depth technical description of the various methods is beyond the scope of this paper and is provided in the literature.

2.1. Embedding

Embedding is a popular technique in deep learning for handling categorical data. It was specifically developed for the use of words, and it is computationally efficient with big datasets [19]. Embedding operates as a trainable layer in which categorical data are encoded as a dense vector of real values with reduced dimensionality. This method is an alternative to one-hot encoding, which is not trainable. Embedding stores the information of n words in an $n \times m$ matrix, where $m \ll n$, each word is stored in a $1 \times m$ vector, whereas one-hot encoding stores each word as a $1 \times n$ vector with all but one entry merely 0 (i.e., a sparse matrix). If one has a dataset of k phrases, where $k \gg n$, this is much more efficient than one-hot encoding. Embedding also enables the analyst to explore words that are similar to each other. (Using distance measures, words that are clustered together have similar meanings.)

2.2. Convolutional Neural Networks

A CNN is a type of neural network used to process inputs that are stored in arrays with fixed dimensions [19–21]. It is a popular method used in deep-learning algorithms and is most frequently applied to images. However, CNNs can also be applied to 1D arrays such as signals, text, and sequences and to three-dimensional (3D) arrays such as videos and volumetric images. Regardless of dimensionality, CNNs are used where there is some spatial or temporal ordering and proximity is significant. A 1D CNN can be combined with an embedding layer to produce better results.

In addition to embedding and classification layers, there are two other types of layers that make up a CNN, convolution and pooling, as shown in Figure 1. The convolution layers are the core of the CNN. This layer essentially applies a filter to a subset of the input at a specific instance of time. The application of the filter creates a weighted linear sum of the input's subset to which the filter is applied. Then, a non-linear function is applied by implementing a rectified linear unit that is applied across the entirety of the height and width of the input. The end result of the application of a single convolutional layer (i.e., a filtering function) and the application of the non-linearity yields a feature map. The output of the CNN is a stack of feature maps, each capturing multiple convolutions of the input and each using a different filter.

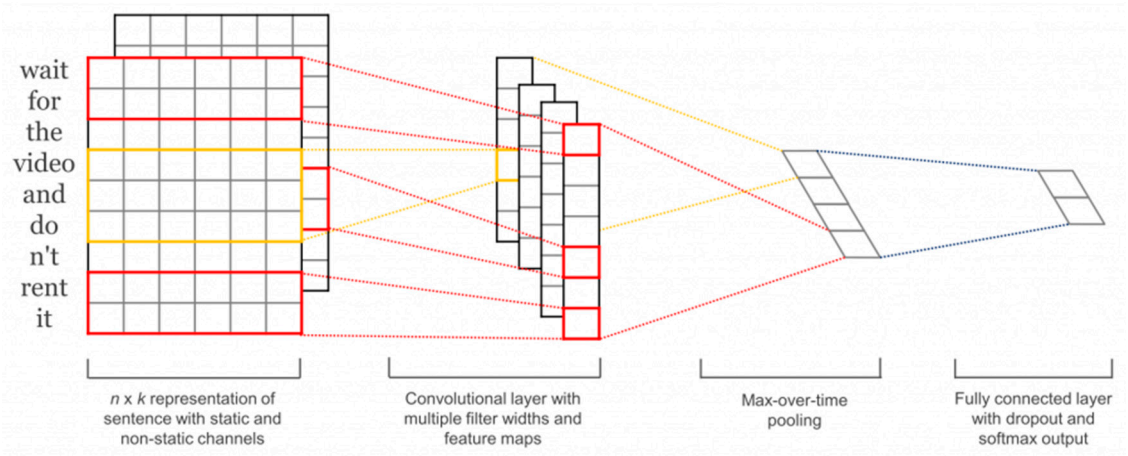

Figure 1. A convolutional neural network for sentence classification [22].

The pooling layers apply non-linear down-sampling functions. For example, one would select the maximum value over non-overlapping subsets of the feature map. These pooling layers are generally applied periodically between convolution layers to reduce the size of the feature map as the computation proceeds through the network. This has three main benefits:

- Reduces the number of parameters for the model,
- Reduces the memory required to perform computations, and
- Reduces overfitting.

Furthermore, the CNN includes fully connected layers that are used to perform classification and regression, and it has regularization techniques that can help to reduce overfitting. One of the most successful regularization methods is called dropout [23]. When training a model using dropout, during each training iteration, a specified percentage of nodes in a given layer and their incoming and outgoing connections are removed at random. Dropout is typically included in CNNs because it improves the accuracy and generalizability of a model by increasing the likelihood that a node will be useful. In addition, spatial dropout [24] is a type of dropout specific to CNNs; it applies dropout to a filter at a layer in the CNN.

2.3. Capsule Networks

CapsNet is a relatively new type of neural network architecture that was first developed in 2017 [25] to address some of the shortcomings of CNNs. These shortcomings are fairly significant and can inhibit a model's generalizability. For example, classic CNNs cannot generalize to new viewpoints. CapsNet attempts to address the Picasso problem, in which a face with all the correct parts but without the correct spatial correlation are recognized as a face. In addition, Sabour et al. [25] claim that capsule networks are pose invariant and can generalize better to new unlearned viewpoints. Lastly, CapsNet could have a stronger resilience to certain types of adversarial attacks.

CapsNet, as shown in Figure 2, removes the pooling layers entirely and replaces them with hierarchical capsule layers. Each capsule applies a subset of the filters applied in the conventional convolutional layer. For example, if 256 filters are applied in the conventional convolutional layer, and there are 32 capsule layers, each capsule layer would be composed of eight filters, each one called a capsule. This is the PrimaryCaps layer. A non-linear weighting function, called a squashing function, is applied to normalize the data and multiplied by a weight. Then, in a process similar to k-means clustering called routing by agreement, the capsules with a similar orientation and magnitude are more heavily weighted in an average across all capsules. This is performed by taking the mean of the capsules, then calculating a weight for each capsule as a function of distance from the mean. A new mean is calculated using those weights, and the weights are recalculated. This is repeated for a set number of times. This is the ClassCaps layer. The squashing function is applied again, and a prediction is made based on the length of the ClassCaps.

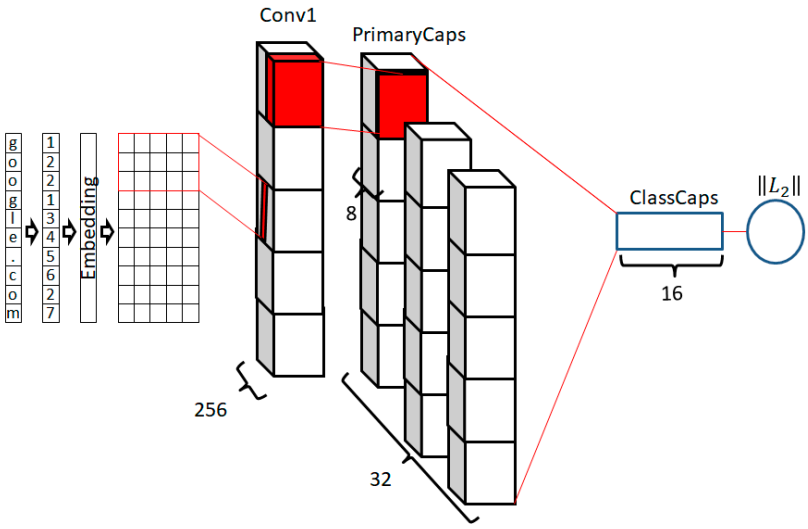

Figure 2. A one-dimensional (1D) capsule network architecture.

2.4. LSTMs

An RNN is a type of neural network capable of receiving input sequences of variable lengths because it processes the inputs one element at a time. An RNN uses the output of the previous input as an additional input for the next element. As a result, RNNs are frequently applied to speech and language problems.

The most commonly used type of RNN is the LSTM unit [26]. Although there are other types of advanced RNNs, the LSTM network is the only one discussed in this paper because of its prominence.

Prior to the introduction of LSTMs, RNNs were difficult to train because the gradients can easily vanish or explode [27]. With the introduction of the LSTM unit, the RNN is easier to train and it is capable of maintaining a long memory. The LSTM approach maintains a "state vector" that contains the "memory" of past events, and this becomes an additional input for the next time step.

Figure 3 shows three RNN models, each with an embedded layer, which are considered herein. In this paper, the LSTM type of RNN is used. The first is a unidirectional RNN, the second is a bidirectional RNN, and the third is a unidirectional RNN with a CNN layer.

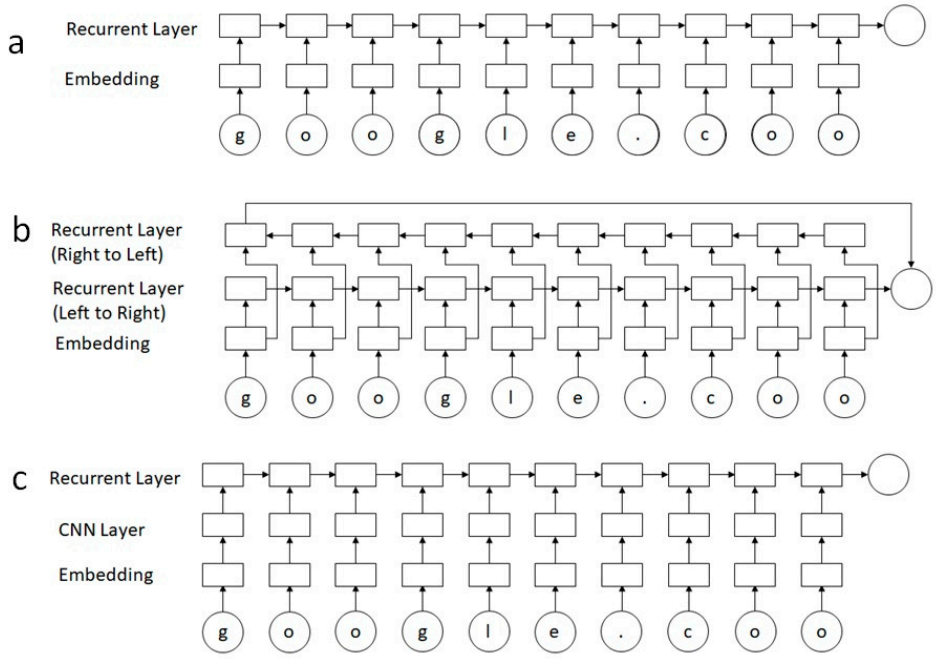

Figure 3. (**a**) A recurrent neural network (RNN) with an embedding layer, (**b**) a bidirectional RNN with an embedding layer, (**c**) an RNN with an embedding layer and a Convolutional Neural Networks (CNN) layer.

3. Model Implementation

All seven models were built and implemented in Python version 3.6.3 using the libraries Tensorflow-gpu version 1.8.0 and Keras version 2.1.6 on four GeForce GTX 1080 Ti graphical processing units (GPUs). Additionally, the metrics were calculated using the functions in the package scikit-learn version 0.20.3. The following models from the literature were built using the descriptions from their respective papers and the parameters specified: LSTM, CNN+LSTM, bidirectional LSTM, and a basic 1D CNN. If certain parameters were not provided, as was the case for the bidirectional LSTM and CNN+LSTM models, the parameters that produced the best results were chosen so that the comparison would be fair. Table 1 provides the parameters of the seven different deep learning models.

The CapsNet code in [28] is based on a two-dimensional (2D) implementation, and was adapted for this study to make use of only 1D data.

All of the models were optimized with the Adam algorithm [29] and trained with a learning rate of 0.001. The models were trained until the validation accuracy showed no signs of increased improvement in batches of 10 epochs.

The domain names were processed without any modification to the characters, and any capitalization was allowed if they existed in the dataset.

Table 1. Table of model parameters and layers.

Model	Batch Size	Model Parameters and Layers
1D Shallow CNN	256	Embedding(128) Conv2D(filters=1000, kernel_size=2, padding='same', kernel_initializer ='glorot_normal') Dropout(0.5) Flatten() Dense(100, kernel_initializer= 'glorot_normal') Dense(1, kernel_initializer='glorot_normal')
1D CNN with MaxPooling	32	Embedding(50) Dropout(0.25) Conv1D(filters=250, kernel_size=4, padding='same') MaxPooling1D(pool_size=3) Conv1D(filters=300, kernel_size=3, padding='same') Flatten() BatchNormalization() Dense(300) Dropout(0.2) BatchNormalization() Dense(1)
Long Short-Term Memory (LSTM)	256	Embedding(128) LSTM(128) Dropout(0.5) Dense(1)
Bidirectional LSTM	256	Embedding(50) Bidirectional(LSTM(128, dropout=0.2, recurrent_dropout=0.2)) Dropout(0.5) Dense(1)
CNN+LSTM	256	Embedding(128) Conv1D(filters=250, kernel_size=4, padding='same') LSTM(128) Dropout(0.5) Dense(1)
Parallel CNNs	256	def conv1DLayer(filter, kernel_size): • Conv1D(filters=filter, kernel_size=kernel_size, input_shape=(maxlen,32), padding='same', activation='relu', strides=1) • BatchNormalization() • Lambda(lambda x:K.sum(x, axis=1),output_shape=(filter,)) • Dropout(.5) x1 = Embedding(32) [conv1DLayer(2,256), conv1DLayer(3,256), conv1DLayer(4,256), conv1DLayer(5,256)] Dense(1024, activation='relu') Dropout(0.5) Dense(1024, activation='relu') Dropout(0.5) Dense(1, activation='sigmoid')
Capsule Network (CapsNet)	318	Embedding(128) Conv1D(filters=256, kernel_size=8, padding='valid') SpatialDropout1D(0.2) Conv1D(filters=512, kernel_size=4, padding='valid') Dropout(0.7) Conv1D(filters=256, kernel_size=4, padding='valid') PrimaryCaps(dim_capsule=8, n_channels=32, kernel_size=4, strides=2, padding='valid') CapsuleLayer(num_capsule=1, dim_capsule=16, routing=7) Length(0.85, 0.15)

4. Datasets

The datasets used for these experiments were generated from two sources:

- The Alexa top one million domains, which formed the list of benign domain names [30].
- The Open-Source Intelligence (OSINT) DGA feed from Bambenek Consulting, which provided the malicious domain names [31]. This data feed was based on 50 DGA algorithms that together contained 852,116 malicious domain names. The dataset was downloaded on May 23, 2018 and DGAs were generated on that day. Also, on April 18, 2019, an additional dataset of 855,197 DGA generated domains was downloaded from OSINT for testing differences in model performance based on time and is regarded as a separate test dataset.

Hence, the resulting dataset contained 1,852,116 domain names; one million were benign and 852,116 were malicious. This dataset contains two overarching types of DGAs: ones that produce random looking domains with high character entropy (e.g., oxufyrqcqopty.net) ones with low character entropy composed of real words (e.g., addressblamescore.com). There were seven DGAs that used real words or websites in their generation of domain names: *cryptowall*, *gozi*, *matsnu*, *pizd*, *suppobox*, *unknowndropper,* and *Volatile Cedar/Explosive*.

This study considered three different experiments. The first, called random assignment, used the full dataset to create the training, validation, and test data using a 60, 20, and 20% split, respectively. For this experiment, 46.03% of the domain names in the test dataset were malicious.

The second experiment takes the April 18, 2019 dataset downloaded from OSINT and tests the models trained in experiment one. This is done to test the ability of the models to detect real word-based DGAs after a change in the words used to generate these domain names.

The third experiment, called novel DGA, was constructed to test how well the models can generalize to new previously unforeseen (or novel) DGAs. In this experiment, the malicious domain names for the training and validation datasets were created with only 41 of the 50 DGAs, whereas the malicious domain names for the testing dataset came from the remaining nine DGAs. (The 41 DGAs were selected at random.) Again, the benign domain names were assigned to the training, validation, and test datasets using a 60, 20, and 20% split, respectively. In this experiment, 27.35% of the domain names in the test dataset were malicious. It is expected that performance on the test set will be lower in experiment two than in experiment one because the test dataset contains domains from DGAs the model was never trained on. All seven models that were built were trained and tested on these datasets.

The test datasets in both the randomly assigned and the novel DGA experiments contain both types of DGAs. The random assignment dataset contains domain names from all of the real word-based DGAs without considering their date of generation. The novel DGA experiment contains only one of these DGAs: *unknowndropper*.

5. Evaluation Metrics

Four papers reported different metrics [10,11,13,14]. The most commonly used metric was the Area under the [Receiver Operating Characteristic (ROC)] curve (AUC). The AUC is the area under a curve of the false positive rate vs true positive rate for various threshold values. However, because the AUCs differed by such small values, differences could be a result of statistical variation. Therefore, this research considered the partial AUC, up to a false positive rate of 0.1%. A maximum false positive rate of 0.1% was selected because any higher would make the model unusable in a real environment. Additionally, this study considered five other different metrics: accuracy, recall, precision, false positive rate, and F1-score.

All five evaluation metrics are derived from the four metrics found in the confusion matrix, which is based on the calculated prediction versus the ground truth, as shown in Table 2:

Table 2. Results of random assignment experiment with the test dataset.

		Predicted Class	
		Malicious	**Benign**
Actual Class (Ground Truth):	**Malicious**	True Positive (TP)	False Negative (FN)
	Benign	False Positive (FP)	True Negative (TN)

Accuracy (*acc*): The fraction of correctly classified examples. The usefulness of accuracy is limited because there is significantly more risk with misclassifying a malicious domain name than with misclassifying a benign domain name. However, it does provide useful insight when the classes are balanced.

$$acc = \frac{TP + TN}{TP + TN + FP + FN}$$

Recall (r): The fraction of malicious domains that are classified as malicious. This is also called true positive rate (TPR).

$$r = \frac{TP}{TP + FN}$$

Precision (p): The fraction of domains classified as malicious that are actually malicious.

$$p = \frac{TP}{TP + FP}$$

False Positive Rate (FPR): The fraction of benign domains classified as malicious.

$$FPR = \frac{FP}{TN + FP}$$

F1-Score (F_1): The F1-score is the harmonic mean of precision (p) and the true positive rate (r).

$$F_1 = \frac{2}{\frac{1}{r} + \frac{1}{p}} = 2\frac{p * r}{p + r}$$

In addition, the amount of time required to classify one domain name will be determined to consider the efficiency of the models.

6. Results

The models that were trained the fastest were the CNN models, followed by the LSTM models, then the CapsNet model, which took nearly twice as long to train as the LSTM models. However, a single training epoch took no more than 15 minutes. Tables 3 and 4 show the simulation results for the two different experiments. Table 5 shows the breakdown of the accuracy in Table 3 for each DGA and benign data. Table 6 shows the performance of the same models on the time split data collected nearly one year later. Table 7 shows the breakdown of the accuracy in Table 4 for each DGA and benign data. The best two methods for each metric are shown in bold. These tables show that although the CNN+LSTM and CapsNet models provided excellent performance, the other methods' performance was nearly as good. Furthermore, the performance for the random assignment experiment was noticeably improved over that for the novel DGA experiment; however, the results with the novel DGA experiment are still highly accurate. In the random assignment data experiment, both the CNN+LSTM and the CapsNet models were in the top two best performing models for four of the metrics. The CapsNet model had the second best the F1-score and the second worst performance in the partial AUC metric. Additionally, the CapsNet model had the second highest accuracy averaged across all the different types of DGAs and benign data. Performance was even better in the novel DGA experiment, in which the CapsNet model achieved the highest accuracy, F1-score, and recall, and the

second highest partial AUC. It also had the highest accuracy averaged across all the different types of DGAs and benign data.

The hardest DGAs to detect were the real word-based DGAs. In Table 5, the DGAs with the lowest performance were *cryptowall*, *gozi*, *matsnu*, and *virut*. Three of those are real word-based DGAs. Detection of *Volatile Cedar/Explosive*, *pizd*, *suppobox*, and *unknowndropper* varied depending on the model. Table 6 shows that the performance of all the models dropped significantly in detecting *pizd* and *suppobox* after the year. The reason for this is likely that the words used to create these domain names changed, creating vulnerability. Only one of the real word-based DGAs was present in the test dataset for the novel DGA experiment: *unknowndropper*, which was entirely undetected. The difficulty in detecting this real word-based DGA exemplifies how difficult it is to detect novel real word-based DGAs and real word-based DGAs when the words used to generate the domain name change.

Table 8 shows the evaluation times of the various models on a single domain. The times were calculated by dividing the amount of time it took to classify each domain in the random assignment test dataset individually by the number of domains. These results are in line with what would be expected. The fastest methods are the two 1D CNN models, followed by CapsNet and lastly the LSTM models. The LSTM models take an order of magnitude longer to process a single domain, making them the most computationally expensive, with the bidirectional LSTM taking the longest because it has to process the domain forward as well as backward.

Table 3. Results of random assignment experiment with the test dataset.

Model	Accuracy	Recall	Precision	FPR	F_1-Score	Partial AUC
Shallow CNN	0.9927	0.9901	0.9940	0.0051	0.9920	0.9556
1D CNN	0.9936	**0.9924**	0.9937	0.0054	0.9931	0.9577
LSTM (128 embedding)	0.9932	0.9917	0.9935	0.0055	0.9926	**0.9659**
Bidirectional LSTM (embedding)	0.9919	0.9893	0.9930	0.0060	0.9912	**0.9670**
Parallel CNNs	0.9884	0.9836	0.9913	0.0074	0.9874	0.9341
CNN+LSTM	**0.9942**	**0.9930**	**0.9945**	**0.0047**	**0.9937**	0.9626
CapsNet	**0.9938**	0.9916	**0.9950**	**0.0042**	**0.9933**	0.9481

Table 4. Results of the novel domain generation algorithm (DGA) experiment with the test dataset.

Model	Accuracy	Recall	Precision	FPR	F_1-Score	Partial AUC
Shallow CNN	0.9613	0.8695	0.9875	0.0042	0.9247	0.8119
1D CNN	0.9684	0.8964	0.9868	0.0045	0.9394	0.8226
LSTM (128 embedding)	0.9689	0.8949	**0.9904**	**0.0033**	0.9402	0.8214
Bidirectional LSTM (embedding)	0.9689	0.8990	0.9862	0.0047	0.9406	0.8298
Parallel CNNs	0.9585	0.8655	0.9806	0.0065	0.9194	0.7972
CNN+LSTM	**0.9710**	**0.9013**	**0.9917**	**0.0028**	**0.9444**	**0.8535**
CapsNet	**0.9714**	**0.9044**	0.9900	0.0034	**0.9453**	**0.8403**

Table 5. Accuracy of random assignment experiment with the test dataset for benign and DGA type.

Type of Data	CNN+LSTM	1D CNN	Shallow CNN	CapsNet	LSTM	Bidirectional LSTM	Parallel CNNs
Benign	**0.9953**	0.9946	0.9949	**0.9958**	0.9949	0.9940	0.9926
Cryptolocker	**0.9917**	**0.9917**	0.9875	0.9900	0.9883	**0.9933**	0.9817
P2P Gameover Zeus	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Post Tovar GOZ	0.9999	**1.0000**	0.9999	**1.0000**	0.9998	0.9999	0.9999
Volatile Cedar/Explosive	0.9848	0.9747	0.9848	**1.0000**	**0.9899**	0.9848	0.9343
bamital	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
banjori	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
bedep	**1.0000**	0.9394	**1.0000**	**1.0000**	0.9697	0.9697	0.9394
beebone	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
chinad	0.9966	**1.0000**	0.9761	0.9966	**1.0000**	**1.0000**	0.9795
corebot	**1.0000**	**1.0000**	0.9851	**1.0000**	**1.0000**	**1.0000**	**1.0000**
cryptowall	**0.2727**	**0.2727**	0.2273	**0.2727**	**0.3182**	0.2273	0.2273
dircrypt	**0.9792**	**0.9792**	0.9583	0.9722	**0.9792**	**0.9792**	0.9722
dyre	0.9994	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
fobber	0.9779	0.9779	**0.9853**	0.9779	0.9779	**0.9853**	0.9632
geodo	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
gozi	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
hesperbot	**0.9487**	**0.9487**	**0.9487**	0.9231	0.8974	0.9231	0.8718
kraken	**0.9854**	0.9769	0.9679	**0.9831**	0.9730	0.9730	0.9651
locky	**0.9815**	0.9760	0.9766	**0.9834**	0.9699	0.9656	0.9477
madmax	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
matsnu	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
murofet	**0.9988**	0.9974	0.9976	**0.9985**	0.9974	0.9979	0.9954
necurs	**0.9818**	**0.9810**	0.9715	0.9736	0.9756	0.9729	0.9539
nymaim	**0.8998**	**0.8875**	0.8662	0.8818	0.8621	0.8612	0.8342
padcrypt	0.9658	0.9487	0.9402	**0.9658**	0.9316	**0.9658**	0.8632
pandabanker	0.8750	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.8750	0.5000
pizd	**0.9464**	**0.9196**	0.8214	**0.9196**	0.8482	0.4821	0.0804
proslikefan	**0.8980**	**0.8912**	0.8707	**0.8912**	0.8299	0.8299	0.7823
pushdo	0.9178	**0.9348**	**0.9207**	0.8980	0.9008	0.8924	0.8499
pykspa	**0.9613**	**0.9592**	0.9462	0.9547	0.9469	0.9431	0.9290
qakbot	**0.9892**	**0.9884**	0.9834	0.9863	0.9850	0.9868	0.9813
ramdo	**0.9974**	0.9922	**0.9974**	0.9897	**0.9974**	0.9948	0.9716
ramnit	**0.9804**	**0.9736**	0.9699	0.9728	0.9719	0.9726	0.9571
ranbyus	0.9981	0.9981	**0.9983**	0.9971	**0.9985**	0.9981	0.9967
shifu	0.9533	**0.9618**	0.9278	0.9469	0.9384	**0.9554**	0.9618
shiotob/urlzone/bebloh	0.9861	**0.9921**	**0.9905**	0.9885	0.9869	0.9881	0.9711
simda	**0.9895**	**0.9959**	0.9817	0.9804	0.9841	0.9878	0.9414
sisron	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**

Table 5. *Cont.*

Type of Data	CNN+LSTM	1D CNN	Shallow CNN	CapsNet	LSTM	Bidirectional LSTM	Parallel CNNs
Sphinx	**0.9938**	**0.9938**	0.9875	**0.9938**	**0.9938**	0.9875	0.9813
Suppobox	**0.9188**	**0.9086**	0.6142	0.8782	0.6193	0.1980	0.0609
symmi	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9896
tempedreve	**0.8696**	**0.8478**	0.8043	**0.8478**	**0.8478**	**0.8478**	**0.8478**
tinba	0.9936	**0.9940**	**0.9948**	0.9914	0.9916	0.9914	0.9877
unknowndropper	0.9167	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
unknownjs	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9231	0.9487	0.8718
vawtrak	**0.8517**	0.8485	0.8006	**0.9075**	**0.8517**	0.7703	0.5550
Vidro	**0.9767**	**0.9535**	**0.9535**	0.9302	0.9070	0.9302	**0.9535**
virut	0.3248	**0.3419**	0.2393	**0.3419**	0.1624	0.1368	0.1709
Micro Average	0.9040	**0.9049**	0.8892	**0.9047**	0.8880	0.8676	0.8319

Table 6. Accuracy of time split experiment with the test dataset for benign and DGA type.

Type of Data	CNN+LSTM	1D CNN	Shallow CNN	CapsNet	LSTM	Bidirectional LSTM	Parallel CNNs
Benign	**0.9912**	0.9878	0.9868	0.9878	0.9860	**0.9903**	0.9797
Cryptolocker	0.9995	0.9995	0.9975	**1.0000**	0.9990	**1.0000**	0.9990
P2P Gameover Zeus	0.9999	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9998
Post Tovar GOZ	**0.9950**	0.9859	**0.9950**	**0.9990**	0.9940	0.9869	0.9789
Volatile Cedar/Explosive	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Bamital	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Banjori	0.9857	0.9857	0.9714	**0.9943**	0.9800	**0.9886**	0.9629
Bedep	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Beebone	0.9889	0.9941	0.9915	**0.9967**	0.9941	**0.9954**	0.9759
Chinad	**1.0000**	0.9964	**1.0000**	0.9964	0.9964	0.9964	0.9893
corebot	0.2447	0.2021	**0.3085**	**0.2660**	0.2234	0.1915	0.1809
cryptowall	**0.9875**	0.9806	0.9806	**0.9889**	0.9764	0.9722	0.9528
Dircrypt	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	0.9999
dyre	**0.9950**	0.9833	0.9900	**0.9917**	0.9900	0.9833	0.9700
Fobber	0.9965	**1.0000**	**1.0000**	**1.0000**	0.9983	0.9965	0.9983
geodo	**0.0833**	0.0417	0.0417	0.0000	**0.0833**	0.0417	0.0417
Gozi	**0.9479**	**0.9479**	**0.9479**	**0.9531**	0.9271	0.9271	0.8750
hesperbot	**0.9933**	0.9851	0.9818	**0.9923**	0.9831	0.9817	0.9676
Kraken	0.9636	**0.9644**	0.9512	0.9619	**0.9651**	0.9621	0.9392
locky	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Madmax	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
matsnu	**0.9983**	0.9969	0.9959	**0.9975**	0.9974	**0.9975**	0.9954
Murofet	**0.9818**	**0.9837**	0.9774	0.9761	0.9754	0.9739	0.9499

Table 6. *Cont.*

Type of Data	CNN+LSTM	1D CNN	Shallow CNN	CapsNet	LSTM	Bidirectional LSTM	Parallel CNNs
Necurs	**0.8930**	0.8758	0.8590	**0.8762**	0.8615	0.8625	0.8097
nymaim	**0.9670**	0.9479	0.9427	0.9444	**0.9514**	0.9462	0.8490
padcrypt	0.9091	**0.9697**	**0.9697**	0.8788	**1.0000**	0.9394	0.8485
pandabanker	**0.4686**	**0.5784**	0.4451	0.4471	0.4490	0.1882	0.0745
Pizd	**0.8808**	**0.8577**	0.8269	0.8423	0.8513	0.8372	0.7449
proslikefan	**0.9208**	**0.9256**	0.9060	0.8815	0.8869	0.8869	0.8655
pushdo	**0.9635**	0.9577	0.9490	**0.9610**	0.9514	0.9430	0.9063
Pykspa	**0.9893**	**0.9876**	0.9831	0.9861	0.9855	0.9874	0.9767
Qakbot	0.9985	0.9965	**0.9995**	0.9970	**0.9995**	0.9955	0.9910
Ramdo	**0.9861**	0.9792	0.9815	**0.9874**	0.9769	0.9742	0.9521
Ramnit	**0.9982**	0.9979	**0.9983**	0.9977	0.9977	0.9980	0.9958
ranbyus	0.9717	**0.9743**	0.9648	**0.9773**	0.9605	0.9515	0.9369
shifu	0.9929	0.9944	**0.9950**	**0.9955**	0.9916	0.9867	0.9703
shiotob/urlzone/bebloh	**0.9942**	**0.9973**	0.9861	0.9919	0.9857	0.9836	0.9364
Simda	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
Sisron	**0.9869**	0.9804	**0.9869**	**0.9869**	**0.9896**	0.9856	0.9817
Sphinx	0.0483	**0.1016**	0.0375	0.0582	**0.0621**	0.0325	0.0276
Suppobox	0.9375	**0.9688**	0.8750	0.9375	0.9375	**0.9844**	0.8594
symmi	**0.9277**	0.8795	0.8795	**0.9317**	0.8675	0.8554	0.8153
tempedreve	**0.9964**	0.9953	**0.9974**	0.9961	0.9940	0.9926	0.9880
tinba	0.9167	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
unknowndropper	**0.8944**	0.8611	0.8444	0.8333	**0.9056**	0.8333	0.7556
unknownjs	**0.8962**	0.8730	0.8606	**0.9508**	0.8711	0.7511	0.5667
Vawtrak	0.9100	**0.9200**	**0.9200**	0.9100	0.9100	0.9000	0.8600
Vidro	**0.3633**	0.3083	0.2217	**0.3100**	0.1767	0.1433	0.1150
virut	**0.8742**	**0.8742**	0.8656	0.8704	0.8673	0.8530	0.8246
Micro Average	**0.9912**	0.9878	0.9868	0.9878	0.9860	**0.9903**	0.9797

Table 7. Results of novel DGA experiment with the test dataset for benign and DGA type.

Type of Data	CNN+LSTM	1D CNN	Shallow CNN	CapsNet	LSTM	Bidirectional LSTM	Parallel CNNs
Benign	**0.9972**	0.9955	0.9958	**0.9966**	0.9958	0.9955	0.9935
dircrypt	**0.9653**	0.9458	0.9347	0.9569	0.9486	**0.9597**	0.9236
hesperbot	**0.8698**	0.8594	0.8333	**0.8906**	0.8594	0.8385	0.7500
pandabanker	0.5455	**0.8485**	0.7879	**1.0000**	0.5455	0.6667	0.5455
proslikefan	**0.8141**	0.8103	0.7590	0.7923	0.7987	**0.8167**	0.7269
pykspa	0.8127	0.8074	0.7695	**0.8303**	**0.8167**	0.8028	0.8015
ramnit	**0.9548**	0.9492	0.9276	0.9525	0.9508	**0.9536**	0.9135
sisron	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**	**1.0000**
unknowndropper	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**
vawtrak	0.3756	**0.3848**	0.3152	**0.4137**	0.3994	0.3857	0.3448
Micro Average	0.7335	**0.7601**	0.7323	**0.7833**	0.7315	0.7419	0.7000

Table 8. Evaluation time (in ms) of a single domain.

Model	Evaluation Time
Shallow CNN	1.95
1D CNN	1.82
LSTM (128 embedding)	21.58
Bidirectional LSTM (embedding)	60.16
Parallel CNNs	4.06
CNN+LSTM	25.88
CapsNet	2.86

7. Conclusions

DGAs play a significant role in a variety of cyber-attacks, and their seemingly random nature makes machine-learning techniques an essential tool in detecting these attacks. Through a thorough investigation of various types of deep-learning models and evaluation criteria, this study identified several supervised learning models that are highly effective at detecting malicious domain names generated by DGAs. In particular, CapsNet and CNN+LSTM performed very well, with CNN+LSTM performing better in the randomly assigned data experiment, and CapsNet performing better in the novel DGA experiment. The CapsNet model is also an order of magnitude faster in making predictions. However, the CapsNet model took longer to train. All models' performances are only slightly degraded when they encounter novel DGAs. These results show that CapsNet can be used by an IT security team to perform real-time DGA detection, as it combines the speed of the CNN models with the accuracy of the CNN+LSTM model, even if the CapsNet is slightly slower to train. More generally, the CapsNet architecture can perform well in 1D applications, not just 2D image classification applications, lending itself to a variety of NLP problems. Finally, because CNNs have a variety of uses in the cybersecurity space [32], and CapsNets can be applied in any instance CNNs are used, this opens a whole new space of potential applications for CapsNets in the realm of cybersecurity.

The greatest weakness of all the models tested is their deficiencies in detecting real word-based DGAs. In some cases, some of these real word-based DGAs use a limited dictionary to generate domain names and change that dictionary after some time. This manifests in three ways. The first is that when the model is trained on data from that DGA, time is not taken into account and the model fails to detect the malicious domain names, as is the case for *matsnu* and *gozi*. The second is when the model can only detect the malicious domain names when it is trained on data from that DGA, regardless of time, but fails to detect it otherwise, as is the case for *unknowndropper*. Finally, there are models that initially perform well but after time passes, performance significantly declines because of a change in the DGA generator, as is the case with *pizd* and *suppobox*. Developing a model capable of detecting malicious domains in all three of these situations is critical, and all models tested here fail to do so. Therefore, future work will focus on these real word-based DGAs.

The most significant limitation in this study is that the benign domains in the training, validation, and test datasets are only from the Alexa top 1 million dataset, which does not necessarily include benign domains from ad networks. These domains might look more similar to some DGA domains. However, producing this dataset was outside scope of this work and should such a dataset be made publicly available, it would be useful to evaluate these models against it.

Funding: This research received no external funding.

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

References

1. Domain Generation Algorithm (DGA). Available online: https://searchsecurity.techtarget.com/definition/domain-generation-algorithm-DGA (accessed on 23 April 2019).
2. McGrath, D.K.; Gupta, M. Behind Phishing: An Examination of Phisher Modi Operandi. In Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats, LEET '08, San Francisco, CA, USA, 15 April 2008.
3. Bilge, L.; Kirda, E.; Kruegel, C.; Balduzzi, M. EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis. In Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, CA, USA, 6–9 February 2011.
4. 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.
5. Yadav, S.; Reddy, A.K.K.; Reddy, A.L.N.; Ranjan, S. Detecting algorithmically generated domain-flux attacks with DNS traffic analysis. *IEEE/ACM Trans. Netw.* **2012**, *20*, 1663–1677. [CrossRef]
6. Antonakakis, M.; Perdisci, R.; Nadji, Y.; Vasiloglou, N.; Abu-Nimeh, S.; Lee, W.; Dagon, D. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware. In Proceedings of the 21st USENIX Security Symposium, Bellevue, WA, USA, 8–10 August 2012.
7. Nhauo, D.; Sung-Ryul, K. Classification of malicious domain names using support vector machine and bi-gram method. *J. Secur. Appl.* **2013**, *7*, 51–58.
8. Demertzis, K.; Iliadis, L. Evolving smart URL filter in a zone-based policy firewall for detecting algorithmically generated malicious domains. In *International Symposium on Statistical Learning and Data Sciences*; Springer: Cham, Switzerland, 2015; pp. 223–233.
9. Shibahara, T.; Yamanishi, K.; Takata, Y.; Chiba, D.; Akiyama, M.; Yagi, T.; Ohsita, Y.; Murata, M. Malicious URL sequence detection using event de-noising convolutional neural network. In Proceedings of the 2017 IEEE International Conference on Communications, Paris, France, 21–25 May 2017; pp. 1–7.
10. Woodbridge, J.; Anderson, H.S.; Ahuja, A.; Grant, D. Predicting domain generation algorithms with long short-term memory networks. *arXiv preprint* **2016**, arXiv:1611.00791.
11. Mac, H.; Tran, D.; Tong, V.; Nguyen, L.G.; Tran, H.A. DGA Botnet Detection Using Supervised Learning Methods. In Proceedings of the Eighth Symposium on Information and Communication Technology (SoICT 2017), Nha Trang, Vietnam, 7–8 December 2017; pp. 211–218.
12. Lison, P.; Mavroeidis, V. Automatic Detection of Malware-Generated Domains with Recurrent Neural Models. *arXiv preprint* **2017**, arXiv:1709.07102.
13. Yu, B.; Gray, D.L.; Pan, J.; De Cock, M.; Nascimento, A.C.A. Inline DGA detection with deep networks. In Proceedings of the 2017 IEEE International Conference on Data Mining Workshops (ICDMW), New Orleans, LA, USA, 18–21 November 2017; pp. 683–692.
14. Zeng, F.; Chang, S.; Wan, X. Classification for DGA-Based Malicious Domain Names with Deep Learning Architectures. *Int. J. Intell. Inf. Syst.* **2017**, *6*, 67–71. [CrossRef]
15. Saxe, J.; Berlin, K. eXpose: A character-level convolutional neural network with embeddings for detecting malicious URLs, file paths and registry keys. *arXiv preprint* **2017**, arXiv:1702.08568.
16. Tran, D.; Mac, H.; Tong, V.; Tran, H.A.; Nguyen, L.G. A LSTM based framework for handling multiclass imbalance in DGA botnet detection. *Neurocomputing* **2018**, *275*, 2401–2413. [CrossRef]
17. Yu, B.; Pan, J.; Hu, J.; Nascimento, A.; De Cock, M. Character level based detection of DGA domain names. In Proceedings of the 2018 International Joint Conference on Neural Networks (IJCNN), Rio, Brazil, 8–13 July 2018; pp. 1–8.
18. Zhao, W.; Ye, J.; Yang, M.; Lei, Z.; Zhang, S.; Zhao, Z. Investigating capsule networks with dynamic routing for text classification. *arXiv preprint* **2018**, arXiv:1804.00538.
19. Bengio, Y.; Ducharme, R.; Vincent, P.; Jauvin, C. A neural probabilistic language model. *J. Mach. Learn. Res.* **2003**, *3*, 1137–1155.
20. LeCun, Y.; Boser, B.E.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.E.; Jackel, L.D. Handwritten digit recognition with a back-propagation network. In Proceedings of the Advances in Neural Information Processing Systems, Denver, CO, USA, 26–29 November 1990; pp. 396–404.

21. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
22. Kim, Y. Convolutional neural networks for sentence classification. *arXiv* **2014**, arXiv:1408.5882.
23. Srivastava, N.; Hinton, G.E.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
24. Tompson, J.; Goroshin, R.; Jain, A.; LeCun, Y.; Bregler, C. Efficient object localization using convolutional networks. In Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 648–656.
25. Sabour, S.; Frosst, N.; Hinton, G.E. Dynamic routing between capsules. In Proceedings of the Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, 4–9 December 2017; pp. 3856–3866.
26. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
27. Bengio, Y.; Simard, P.; Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **1994**, *5*, 157–166. [CrossRef] [PubMed]
28. CapsNet-Keras. Available online: https://github.com/XifengGuo/CapsNet-Keras (accessed on 24 April 2019).
29. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv preprint* **2014**, arXiv:1412.6980.
30. Does Alexa have a list of its top-ranked websites? Available online: https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites (accessed on 25 May 2018).
31. Bambenek Consulting—Master Feeds. Available online: http://osint.bambenekconsulting.com/feeds/ (accessed on 22 May 2018).
32. Berman, D.S.; Buczak, A.L.; Chavis, J.S.; Corbett, C.L. A survey of Deep Learning Methods for Cyber Security. *Information* **2019**, *10*, 122. [CrossRef]