LITERATURE REVIEW

There are numerous studies that indicate the use of machine learning for fingerprinting IoT devices. Multiple works have developed models that accurately classify digital devices into categories of IoT and Non-IoT, yet are not wide enough for real-world use. Most prior works utilize service banner text to differentiate between device types. Kumar et al. designed an ensemble of IoT classifiers based on UPnP and DNS responses, HTTP banners, and network information, achieving a 92% coverage and 96% accuracy on 1000 manually labeled devices. Despite this high accuracy, other research studies elected to solely observe network traffic to fingerprint IoT devices. Guo et al. posited that IoT devices can be classified by observing network flow because such devices exchange data with their manufacturer’s servers. After discovering nearly 200 candidate servers accessed by 26 devices across 15 vendors, their methodology successfully identified IoT devices connected across the University of Southern California (USC). Meidan again tested Guo’s postulate, using a supervised algorithm to classify manually labeled IoT devices in a localized lab environment to extract TCP packet features from such devices, including baby monitors, IP cameras, and printers in order to discriminate between IoT and non-IoT devices. Miettinen expounded on Meidan’s work, developing a random forest classifier trained on data from IoT device set up, allowing for the capture of device specific traits and mapping of such traits to device type. Improving anomaly and infection detection is reliant on distinguishing between device types. Nguyen et al utilized machine learning classification capabilities to not only discriminate between IoT and non-IoT devices, but also to detect anomalies in IoT devices, creating rapid intrusion detection at a high accuracy when trained on local networks. Thangavelu extends this idea to an Internet Service Provider’s (ISP) perspective, building a large-scale machine learning model capable of overcoming the limitations of past centralized approaches.

Recently, Pinheiro introduced the novel technique of distinguishing between devices based on outgoing packet specifications. He utilized the feature statistics of the packet flows studied, combining the mean and standard deviation of packet lengths with the number of bytes sent by each device in one second intervals. Siby et al furthers by passively intercepting wireless signals in local networks to extract MAC addresses from flows allowing for IoT device identification. Alternatively, Acar et al developed web scripting that identifies the presence of IoT devices running local HTTP services, disclosing vulnerabilities to the user of the script.

Safei Pour et al identify a shortcoming in the mentioned literature—the scope of prior work is limited to local networks and as such, does not scale to a full internet-wide perspective. They instead leverage Internet-wide network traffic to develop deep learning techniques to identify unreachable infected devices and predict their type from the features extracted from TCP SYN packets. While their classifier is highly accurate compared to the large expanse of the Internet, it is still lower compared to models based on local networks. Yet with all these recent developments, there is no preceding work directly comparing each of these methods against each other on an Internet-wide scale. In contrast to past works, a focus will be placed on direct comparison on past methods rather than solely on one new method. This work aims to fill the gap in the knowledge of internet wide IoT classifiers, providing justification for which model and methodology creates the strongest classifier. The main reason for the importance of such a work is the need for a wide classifier in order to develop targeted security fixes for IoT devices. Creating the capability to classify each infected device on the internet allows for companies to be notified if they house an infected device and gives manufacturers the ability to make rapid target fixes to remove vulnerabilities from IoT devices. While such classifiers do exist, a full comparative study allows for improvements to be made to these classifiers. The other issue with the existing classifier Safei Pour implements is its large runtime constraint. As full monthly optimization to keep the model up to date takes over half a day to run, such a model would not be able to respond to rapidly evolving cyber threats in a reasonable time. As model performance and runtime responds to the data provided, it is important to address the time constraint by focusing on the gap in the current research—whether training on raw packet data improves model performance compared to network data.

METHODOLOGY

The underlying methodology determines similarities in network traffic flows that are exclusive to IoT devices and their respective malware in order to differentiate between malicious IoT and non-IoT devices. In order to focus on differences in the data, five baseline models will be used with comparisons only being made between the same model types. The four models will be from Safei Pour’s work, as their code is made available publicly. The first three models are based off a convolutional neural network architecture, using multiple layers of neurons, backpropagation, and error correction to learn about the data from a given input. The convolutional models use dynamic kernels to extract and pool features from the data. The convolutional models used are 2-dimensional convolutional neural network (2-dimension kernels; 2D-CNN) 1-dimensional convolutional neural network (1-dimension kernels; 1D-CNN) and multi window 1D-CNN (Multiple 1D-CNN's in parallel; MW-1D-CNN). The convolutional models will take the matrix representation of packet flows with packets and features of the packets in a form where and the th packet on the flow is . In these models, convolutional kernels are applied on the input matrix to discover local correlation between the packets in the packet flow for a device and across other devices. The 2D-CNN contains consecutive two-dimensional convolutional layers (with kernels of size followed by max pooling, hidden layers, and a Softmax classifier at the end. In a 1D-CNN, kernels have a width equal to the width of the input matrix (. The MW-1D-CNN concatenates the outputs of varying kernel heights to better extract correlation between the packet features. In return the model’s first layer returns an output where is the sequence of packets , represents the bias, and , the nonlinear activation function. The convolutional filter passing over the data creates a feature map to which max pooling is performed over taking the maximum value of . The use of differing window heights is critical to capturing the varying dynamics of darknet packet flows. The other two models are based off the Random Forest Architecture, one based off the raw packet data, and the other based on the feature statistics of the packets (ie network data) [5]. These proposed convolutional models were implemented in the Keras library (version 2.3.1) with a Tensorflow-gpu backend (version 1.14) in Python 3.7. As there was a large difference between the number of samples of IoT devices and non-IoT devices, cost sensitive learning was implemented. To prevent overfitting of data, the number of epochs was set to a constant 30. In order to optimize the models, Tree Parzen Estimation was used to find the best set of hyperparameters in the search space presented in figure (#) out of 100 trials with respect to loss. Random Forest models were implemented in the Scikit-Learn package and the best model was retrieved using random search on the search space summarized in table (#). Tables # and # are reported using the *Begin:Step:End* format. The convolutional models are trained and evaluated on four NVIDIA GeForce RTX 2080TI GPU’s each with 11 GB of RAM, 4352 CUDA cores, and 544 Tensor Cores to parallelize the process of training. The random forest models are trained on an AMD Threadripper 2990WX with 32 cores and 64 threads supplied with 128GB of RAM. To compare the performance of the models, the standard metrics precision, recall, F-measure, and the Area under the Receiver Operating Characteristic curve (AUC-ROC) were used. Precision is defined as the ratio of correctly classified IoT devices (true positives) and the total number of instances designated as IoT by the model (true positives + false positives) and is represented as . Precision demonstrates the model’s ability to designate only relevant cases as relevant. Recall is defined as the ratio of correctly classified IoT devices and the total number of IoT devices within the dataset (true positives + false negatives) and is represented as . Recall is used to show the model’s ability to find all relevant cases in the dataset. In order to get a better representation of the model’s performance, both of these metrics are brought together using F-measure, which takes the weighted average of precision and recall, and is defined as . AUC-ROC measures the entire two-dimensional area underneath the ROC curve .

The models after optimization will be run 50 times. Accuracy, loss, precision, recall, F-Measure, and AUC-ROC will be collected for each run for comparison. In addition to straight comparison across training on the same data, 20-cross-fold validation will be run on each model to simulate real world performance. The general procedure for 20-cross-fold validation is listed below:

1. Shuffle the dataset randomly
2. Split the dataset into 20 groups
3. For each unique group:
   1. Take one group as test data (hold out group)
   2. Take the remaining groups as the training data
   3. Fit a model on the training data and evaluate on the hold out.
   4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the evaluation score of the held out group.

For clarity, the methods are summarized below.

1. Collect raw data using darknet scanner.
2. Distinguish easily classifiable IoT and non-IoT devices to form train and test datasets.
3. Develop the specified models and perform hyperparameter optimization using the train dataset.
4. With the optimized hyperparameters, train the models with the test dataset and evaluate their performance using the test dataset.
5. Collect and evaluate differences between the metrics, performance, and time cost of the models.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameters | Space | 2D-CNN | 1D-CNN | MW-1D-CNN |
| Batch Size | 256,512,1024 | 1024 | 512 | 1024 |
| Number of Kernels | 32,64,128 | 32 | 32 | 128 |
| Kernel Size | (2,2),(3,3) | (3,3) |  |  |
| Kernel Height | 2,4,8,16,32,64 |  | 64 |  |
| Max Height | 40:10:80 |  |  | 80 |
| Activation Function | Relu, Sigmoid, Tanh | Sigmoid | Tanh | Sigmoid |
| Pool Size | 2,3 | 2 | 3 |  |
| Dropout Rate | U(0.1,0.3) | 0.291 | 0.243 | 0.249 |
| Optimizer | SGD, Adam, RMSprop | RMSprop | Adam | Adam |
| Learning Rate | U(0.0009, 0.00225) | 0.00143 | 0.00122 | 0.00136 |
| CNN Layers | 1:1:4 | 2 | 3 |  |

*Table 1:* Selected Hyperparameters after Optimization

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Model | Time (Hr:M:S) | Loss | Accuracy | AUC-Roc |
| 2d-CNN | 4:40:51 | 0.276 | 0.872 | 0.910 |
| 1D-CNN | 2:59:16 | 0.250 | 0.895 | 0.920 |
| MW-1D-CNN | 19:21:03 | 0.247 | 0.892 | 0.922 |

*Table 2:* Run-Time, Loss, Accuracy, and AUC-ROC score after Optimization

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Precision | Recall | F-Measure |
| 2D-CNN |  |  |  |
| 1D-CNN |  |  |  |
| MW-1D-CNN |  |  |  |

*Table 3:* Precision, Recall, and F-Measure Scores relative to IoT devices



*Figure 1:* AUC-ROC of Optimized 1D-CNN



*Figure 2:* Zoomed in AUC-ROC of Optimized 1D-CNN



*Figure 3:* Precision-Recall Curve of Optimized 1D-CNN

**

*Figure 4:* Weighted Precision-Recall Curve of Optimized 1D-CNN



*Figure 5:* AUC-ROC of Optimized MW-1D-CNN



*Figure 6:* Zoomed in AUC-ROC of Optimized MW-1D-CNN



*Figure 7:* Precision-Recall Curve of Optimized MW-1D-CNN



*Figure 8:* Weighted Precision-Recall Curve of Optimized MW-1D-CNN



*Figure 9:* AUC-ROC of Optimized 2D-CNN



*Figure 10:* Zoomed in AUC-ROC of Optimized 2D-CNN



*Figure 11:* Precision-Recall Curve of Optimized 2D-CNN



*Figure 12:* Weighted Precision-Recall Curve of Optimized 2D-CNN