Example Introduction -  
This study investigated the implicit self-theories secondary students held about intelligence and research abilities. A brief review of the literature illustrated the findings on learned helplessness, causal attributions for failure and success, and self-handicapping behaviors of students. While there was clearly research on self-theories of intelligence, the researcher did not know how his students viewed intelligence or research abilities, and this study was designed to investigate that gap by using an eight-item survey instrument to collect quantitative information about the students’ views. The students attended a large urban high school in South Florida, and were enrolled in a research-based course. Based on the information collected in the survey, students were classified as holding entity theories, incremental theories, or no set theories of both intelligence and research abilities. Paired Samples T-Tests were used to determine whether there was statistical significance in the differences between students’ Implicit Theories of Intelligence Scores (ITIS) and Beliefs About Research Abilities Scores (BARAS). Descriptive statistics were also used to look at the correlations between implicit theories of intelligence and research abilities. The differences in the ITIS and BARAS scores were found to be statistically significant for five of the nine groups of respondents that were tested. The results of the descriptive statistics illustrated that students can be entity theorists about intelligence and incremental theorists about research abilities, as well as incremental theorists about intelligence and entity theorists about research abilities. Suggestions were made by the researcher to use qualitative methods to investigate this phenomenon further.

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 more intensive models improve classification performance of IoT devices while maintaining time efficiency.

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 three models will be from Safei Pour’s work, as their code is made available publicly. The 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 [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 *Table 1* out of 100 trials with respect to loss. *Table 1* is 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. 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 where

The models after optimization will be run 50 times. In order to find the most efficient model with a negligible effect on accuracy, the data supplied from each device will be cut short to number of packets, limiting the data used by the model and thus speeding up its training. Each model has a minimum number of packets needed to train and thus, the number of packets used for each model is listed in *Table 4*. Accuracy from applying the model on the test dataset will be collected for each run for comparison. In addition to compare fixed dataset performance, 20 fold cross validation will be run as the dataset is split randomly, simulating real world performance. The general procedure for 20 fold cross 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 (held-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.

In order to compare model performance across all trials, one-way ANOVA was utilized to determine if there was a statistical difference between the model performances. Two sample T-Tests were utilized for post-hoc testing in between two specific models within the data gathered. P values under 0.01 from ANOVA and T-Tests were marked as statistically significant meaning the chance of correlation due to chance was 1% or less. While a p value of 0.05 is typically used, the chosen p value was used as the variance between model runs was significantly low due to the computational nature of the study. The stochastic nature of these models also necessitates the need for a low p-value to remove any doubt caused by random chance. For clarity, the methods are summarized below.

1. Develop the specified models and perform hyperparameter optimization using the train dataset.
2. With the optimized hyperparameters, train the models with the test dataset cut off at packets per device and evaluate their performance using the test dataset.
   1. This is performed 50 times for each value used.
3. Perform 20-Cross-Fold-Validation on each of the models to evaluate real world performance of the model.
4. Collect and evaluate differences between the metrics, performance, and time cost of the models.

RESULTS

Each of the three models were optimized using 100 trials on a specified search space and determined hyperparameters are shown in *Table 1.1*. Statistics for the optimized runs are shown in *Tables 1.2* and *1.3*. Tests regarding the 20 fold cross validation runs are displayed in *Tables 2.x-2.y*.Tests regarding the packet selection are displayed in *Tables 3.x-3.y*. *Figures 1.1-3.4* demonstrate the ROC and Precision-Recall curves of each of the models. *Figures 4.1-4.3* visually display the relationships between accuracy, run time, and number of packets the model was trained on.

*Run-Time*

Most prominent is the difference in the time to optimize, with the MW-1D-CNN taking 15 hours longer than the 2D-CNN and 16 hours longer than the 1D-CNN. This difference in run time is expected due to the complexity of the models. As all models used have the same linear layers applied after the convolutional output, the difference herein lies in the convolutional layers. The maximum convolutional layers of the sequential models is limited to 4 layers with one convolutional operation per layer while the Multi Window Model while only having 1 convolutional layer has at least 40 convolutional operations running parallel to each other significantly increasing the run time (*Insert T Test Result Here*).

*Model Performance*

Surprisingly, while the Multi Window model is much more complex than the sequential 1D Convolutional model, the sequential 1D model seemingly performs just as well in not only the optimized runs, but in the 20 fold cross validation tests (, , , see *Table 2.2*) and data restriction tests (*Insert T Test Result Here*) as well. This could suggest that the 1D kernels used in both models effectively capture the dynamics of the IoT devices and can confidently discriminate them from non-IoT devices. Compared to the 2D model, both of the 1D based models outperform in classification of IoT devices and discrimination against non-IoT devices (*Insert Both T Tests Here and ANOVA*). This result correlates with the differences in, , and between the optimized models, suggesting the 2D architecture falters at both discriminating against non-IoT devices (more false negatives) and classifying IoT devices (less true positives). The higher loss measure across all packet lengths tested implies that the CNN-2D yields the most deviation between the predicted and actual result (*insert means here*) and is further supported by the lowest accuracy across all tests run (*insert a bunch of means here and T-Test supporting this*)

*Packet Restrictions*

LIMITATIONS

There are some important limitations to be noted in my research. The contributing factor limiting the scope of my work was the 2-week breakdown of the computer I used to train and evaluate the models. This limited the models I could use without crashing the computer limiting the scope of the research to only convolutional models. This possibility of evaluating other models could yield a better model than the 1D Convolutional Network for the classification of IoT devices. Another limitation regarding the physical specifications of the computer is the large power draw and spacing of components leading to high temperatures. The large power consumption of the 2080Ti’s generates lots of heat when training at full capacity. Combined with the cramped spacing of the 4 GPU set up limiting cooling capabilities, internal temperatures of each GPU remained above 70⁰ for periods of up to 17 hours. In order to preserve its long-term functionality, the GPU automatically reduces its performance slightly to lower power draw and stabilize temperature. This reduction in performance could lead to artificially higher results for run times of each model, meaning run time results are less generalizable to running one model at a time with ample time for the GPU’s to cool off. Regarding the model specifically, each model was randomly initialized, to simulate real-world training of the model. While this effectively simulates model performance in the real world, poor initializations could lead to skewed model performance, worsening its performance and increasing variation, leading to misinterpreted results. This was addressed by removing the few runs that initialized poorly and evaluating only the means of good evaluations for each model. Yet, using the same initialization for each model would yield more stable results with less variation, improving the clarity of results.

Computer machine broke for 2 weeks, crashed while running a model.

* Potential overheating of components slowing down run time.
* Random Initialization (Never used the same random seed is both good and bad)
* Power Draw, could have restricted model performance on GPU.

SUGGESTIONS FOR FUTURE RESEARCH

With the analysis of the model results and discussion on the implications of the results, this work has sufficiently filled the gap in the research by using multitudes of comparative techniques to evaluate different IoT classifier model performances. Further, the new understanding generated by this work opens a route for future investigations on IoT classifiers and development of platforms utilizing such classifiers. Due to changing protocols in IoT devices, behaviors are changing rapidly, yet it is unknown whether long term changes can hinder model performance. With a larger time scale, research can focus on building classifiers resistant to such changes, allowing for new cyber threats to be discovered instantly, instead of discovered when the model is updated. Additionally, further research can aim to not only classify by device type, but also classify distinct viruses and cyber threats associated with the device, allowing for the linking of devices to a central threat. For example, a device could be a part of a collection of similarly infected devices controlled by one group, making it urgent to detect in real-time. On top of that, these developments along with the current work lead to the development of threat detection platforms, which aim to alert organizations of distinct cyber threats. Finally, work must be done on improving the accuracy of classifiers to enable agile IoT characterization. Misclassification can lead to a waste of resources on non-IoT devices while allowing larger threats to go without detection, making it imperative to further the sophistication of such classification models. Overall, these findings serve to catalyze further work in this field with the aim of providing a more secure cyberspace in the future.

APPENDIX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 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) | **X** | **X** |
| Kernel Height | 2,4,8,16,32,64 | **X** | 64 | **X** |
| Max Height | 40:10:80 | **X** | **X** | 80 |
| Activation Function | Relu, Sigmoid, Tanh | Sigmoid | Tanh | Sigmoid |
| Pool Size | 2,3 | 2 | 3 | **X** |
| 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 | **X** |

*Table 1.1:* 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 1.2:* Run-Time, Loss, Accuracy, and AUC-ROC score after Optimization

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Precision | Recall | F-Measure |
| 2D-CNN | 0.88 | 0.87 | 0.84 |
| 1D-CNN | 0.89 | 0.90 | 0.88 |
| MW-1D-CNN | 0.89 | 0.89 | 0.89 |

*Table 1.3:* Precision, Recall, and F-Measure Scores relative to IoT devices

|  |  |  |
| --- | --- | --- |
| Statistics | 1D-CNN | MW-1D-CNN |
| Mean | 87.657324 | 88.12240869 |
| Variance | 0.4505833 | 0.608334905 |
| Observations | 20 | 20 |
| Pooled Variance | 0.5294591 |  |
| Hypothesized Mean Difference | 0 |  |
| df | 38 |  |
| t Stat | -2.021229 |  |
| P(T<=t) one-tail | 0.0251708 |  |
| t Critical one-tail | 1.6859545 |  |
| P(T<=t) two-tail | 0.0503415 |  |
| t Critical two-tail | 2.0243942 |  |

*Table 2.1:* Two Sample T-Test between 20 fold cross validation runs of 1D-CNN and MW-1D-CNN

|  |  |  |
| --- | --- | --- |
| Statistics | 2D-CNN | MW-1D-CNN |
| Mean | 86.09908 | 88.12241 |
| Variance | 0.670175 | 0.608335 |
| Observations | 20 | 20 |
| Pooled Variance | 0.639255 |  |
| Hypothesized Mean Difference | 0 |  |
| df | 38 |  |
| t Stat | -8.00256 |  |
| P(T<=t) one-tail | 5.65E-10 |  |
| t Critical one-tail | 1.685954 |  |
| P(T<=t) two-tail | 1.13E-09 |  |
| t Critical two-tail | 2.024394 |  |

*Table 2.2:* Two Sample T-Test between 20 fold cross validation runs of 2D-CNN and MW-1D-CNN



*Figure 1.1:* AUC-ROC of Optimized 1D-CNN



*Figure 1.2:* Zoomed in AUC-ROC of Optimized 1D-CNN



*Figure 1.3:* Precision-Recall Curve of Optimized 1D-CNN

**

*Figure 1.4:* Weighted Precision-Recall Curve of Optimized 1D-CNN



*Figure 2.1:* AUC-ROC of Optimized MW-1D-CNN



*Figure 2.2:* Zoomed in AUC-ROC of Optimized MW-1D-CNN



*Figure 2.3:* Precision-Recall Curve of Optimized MW-1D-CNN



*Figure 2.4:* Weighted Precision-Recall Curve of Optimized MW-1D-CNN



*Figure 3.1:* AUC-ROC of Optimized 2D-CNN



*Figure 3.2:* Zoomed in AUC-ROC of Optimized 2D-CNN



*Figure 3.3:* Precision-Recall Curve of Optimized 2D-CNN



*Figure 3.4:* Weighted Precision-Recall Curve of Optimized 2D-CNN