DETECTION METHOD FOR CLASSIFYING MALICIOUS FIRMWARE

David Noever and Samantha E. Miller Noever

PeopleTec, Inc., 4901 Corporate Drive. NW, Huntsville, AL, USA

ABSTRACT

A malicious firmware update may prove devastating to the embedded devices both that make up the Internet of Things (IoT) and that typically lack the same security verifications now applied to full operating systems. This work converts the binary headers of 40,000 firmware examples from bytes into 1024-pixel thumbnail images to train a deep neural network. The aim is to distinguish benign and malicious variants using modern deep learning methods without needing detailed functional or forensic analysis tools. One outcome of this image conversion enables contact with the vast machine learning literature already applied to handle digit recognition (MNIST). Another result indicates that greater than 90% accurate classifications prove possible using image-based convolutional neural networks (CNN) when combined with transfer learning methods. The envisioned CNN application would intercept firmware updates before their distribution to IoT networks and score their likelihood of containing malicious variants. To explain how the model makes classification decisions, the research applies traditional statistical methods such as both single and ensembles of decision trees with identifiable pixel or byte values that contribute the malicious or benign determination.

KEYWORDS

Neural Networks, Internet of Things, Image Classification, Firmware, MNIST Benchmark.

1. INTRODUCTION

Image classifiers represent a novel approach to abstracting small differences in program executables, particularly for closely matched cases where human or rule-based inspections fail. This research explores the challenge of detecting malicious firmware. We convert a common executable format from raw bytes to decimal (0-15), then scale this identifying image into 256 greyscale pixel values (Figure 1).

This process of transforming compiled bytes to images extends previous breakthroughs in computer vision and promises continued enhancement as more sophisticated deep learning methods advance. For comparison to previous work, one classic benchmark for machine learning is handwriting digit recognition (Modified National Institute of Standards and Technology database, or MNIST) [1-12]. The original digit recognition challenge has since seen widespread generalization to include alphabetic versions [2] in multiple languages [10-12] and multiple unrelated topic areas [13-19] ranging across medical [13], fashion [14], and satellite imagery [17]. A common element of these generalizations has been that small images (either 28x28 or 32x32) [1,7,16] can be addressed with both statistical machine learning (e.g. tree-based algorithms) or deep learning (multi-layer neural networks) [7]. We have recently built many cyber-security challenge datasets for malware and intrusion detection by first assembling the dataset in formats compatible with previous MNIST solutions [17-19], but also adding to the conversation begun by Intel and Microsoft Research to go beyond the signature-based methods of identifying viruses in their STAMINA initiative(Scalable Deep Learning Approach for Malware Classification) [20-21]. Our datasets for malware (V-MNIST) [18] and image-based intrusion detection [19] are starting points for motivating the current approach to map firmware updates [22-23] that are either malicious, hacks, or benign into a similar format. The approach builds on the extensive publication history of mapping integer datasets to images, then applying the power of convolutional neural networks (CNNs) along with other algorithms to compare their ability to detect malicious or rogue firmware updates [24-25]. One motivation for converting the malware to imagery stems from the advanced feature extractions available for performing convolutions on pixel maps. The core mathematical transformation applied in two-dimensional convolutions includes sliding a small weight matrix over the image, performing elementwise multiplication within that sliding window, then finally summing up the results to generate new output pixel layers. Successive layers involving convolutions automate feature extraction and hierarchies of related image parts. A second investigative motivation behind this approach follows from the success already demonstrated by STAMINA for other categories of malware [20-21] but extended here for firmware rather than traditional malware.

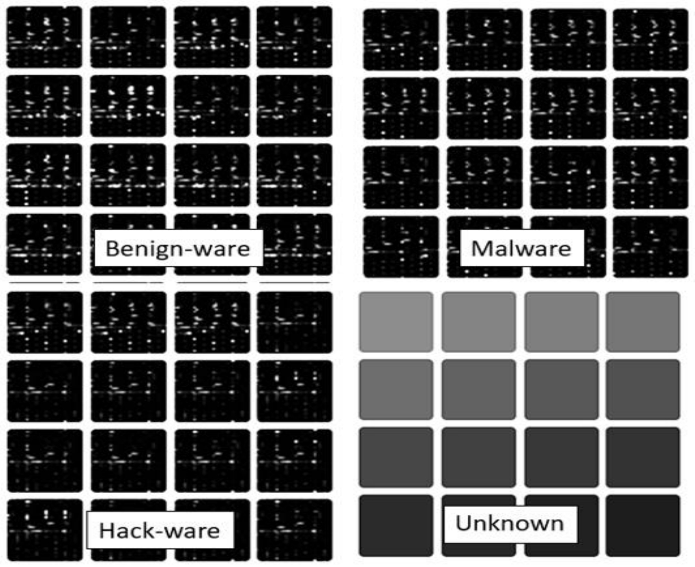


Figure 1. Firmware ELF-binaries as Thumbnail Images

Why the Internet of Things (IoT) firmware? Embedded and Internet of Things (IoT) infrastructure depends on updates that users and industry can trust. What's unclear presently however is whether these updates will prove equally trustworthy given the lackadaisical approach to decent password protection or verifiable software integrity [26-27]. Common security concerns for IoT devices stem from manufacturers who seek convenient installation and configuration methods. These methods often feature automatic attachment to any network (Bluetooth, WiFi, etc.), open serial ports for debugging without any security requirements, and overall a limited set of easily guessable passwords. In 2020, 50 billion IoT devices worldwide are specifically designed to attach to a network with little or no administrative management or oversight [27]. While advanced persistent threats (APTs) have previously exploited weak passwords for devices like thermostats, home appliances, and personal assistants, the infection of firmware updates represents a larger attack surface to exploit. Anecdotal reports from the 2018 Olympics noted that hacked remote printers were unable to issue gate tickets for the opening ceremonies [27-28]. Ideally, a simple image classifier that quickly identifies and isolates rogue firmware might prove useful in the same way that program hashes and signatures defined a previous generation of malware protection layers. The original contribution of this work is to 1) map the firmware labeled dataset to a representative image and 2) solve the classification problem in image-space (pixels) as a proof of principle for future development.

2. METHODS

This research extends the labeled ELF-binary dataset [22] to image classification. We accept the multi-class labels for malware, hack-ware, and benign-ware, which include over 40,000 examples of small compiled binaries. We add class specific to image classification which is grayscale "unknown" and bears no family resemblance to compiled software. The unknowns are just a spectrum of flattened backgrounds shades. The original dataset encodes the binary files using the following annotation and naming scheme:

{Architecture}__{Bit width}__{Endianess}__{ABI}__{Compiler used to compile the exe}__{Optimization level}__{Whether obfuscation was applied}__{Is the file stripped of debug symbols}__{Package name}__{Program name}.

The ELF (Executable Link Format) is widely used in a host of IoT relevant configurations (Figure 2). For operating systems like Linux, Android, BSD, Solaris, and BEOS, the ELF executable consists generally of the header, program header table (text and data). The same structure describes formatted game files (PSP, Playstation 2-4, Dreamcast, Gamecube, and Wii), custom phone operating systems (Nokia, Samsung, Ericsson), and microcontrollers such as Atmel and Texas Instruments.

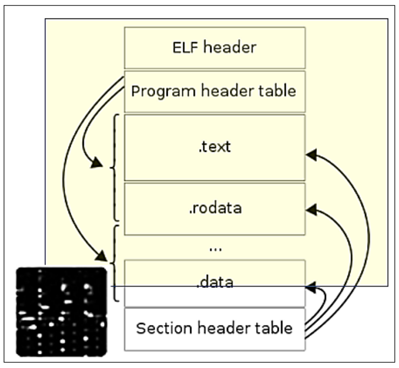


Figure 2. Illustration to map ELF header to images

2.1. Dataset Preparation

Employing similar methods to Project STAMINA from Intel and Microsoft [21], we convert the first 1024 bytes of each firmware binary to its decimal equivalent then scale those integers (0-15) to span the full 0-255 interval to create small images as JPEGs. Because the class imbalances include dominant benign firmware (75% of examples), we produced an alternative public dataset (published on Kaggle [29]) that includes both a long and a short-form version. The short-form version includes 3,000 examples of benign-ware, 714 examples of malware, and 100+ examples of hack-ware. While not balanced, it matches with the presentation of a basic confusion matrix of train-valid-test split. The choice for 1024 bytes as a small thumbnail (32x32 pixels in grayscale) derives from matching this complex problem to previous MNIST approaches but with attention to the stride-length (powers of 2) preferred by some modern deep learning frameworks like Keras. The area of the sliding weight matrix or kernel in 2D convolution determines the number of input features from the firmware that get passed to generate new output features in the deeper layers of the neural network (Figure 3).

2.2. Model Parameters and Quantitative Metrics

As an example of applying deep learning, we solve the firmware-image classifier problem using transfer learning from MobileNetV2 starting networks [30]. This network provides an optimized algorithm for feature hierarchies but efficiently extends to new areas beyond its original training datasets. The network layout features deep convolution operations as intermediate layers, along with an efficient building block approach to adding more operations. This fast, accurate and modular model also introduced a shortcut connection to recall previous layers and training with residuals (rather than gradients) to avoid the flattening of learning curves as gradients vanish near a premature convergence. We have previously found this approach useful to understand the image classification for both malware (V-MNIST) [18] and intrusion detection [19].

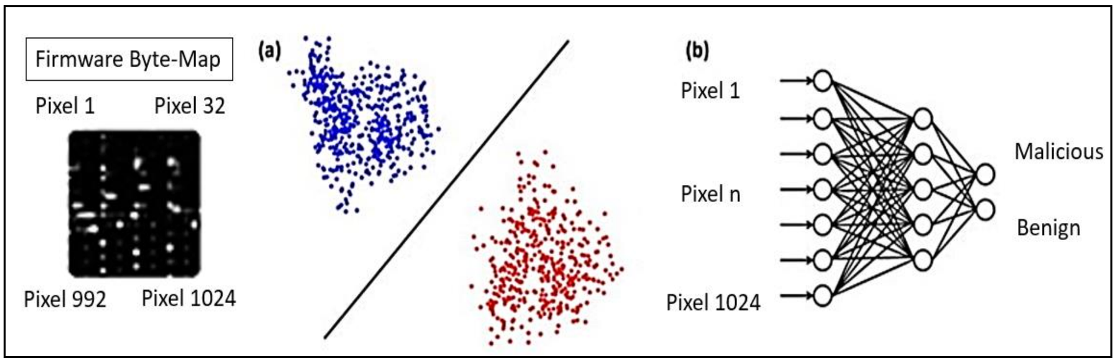
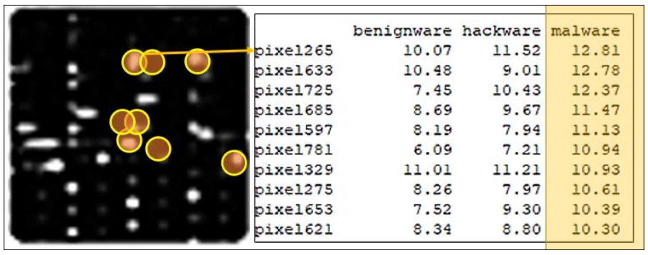


Figure 3. Illustration of steps to map firmware to abstract images (a), then solve the classification problem in image space (b) using both statistical machine learning and deep neural network classifiers.

We use transfer learning with a batch size of 16 for 50 epochs and a 0.001 learning rate, and report four firmware classes: “malware”, “hack-ware”, “benign-ware”, [22] and the new class labeled “unknown”. The unknowns were to handle images outside of the patterns of ELF headers, such as flat grayscale backgrounds. We generate all the images using the ImageMagick tool suite [31] after binary-to-scaled decimal conversions of 1024 pixels, which subsequently rescale to meet the 32x32 requirement. The accuracy and misclassification (via error matrix) provide a score to assess effectiveness. We assess the learning parameters and sample sizes [32] using error and accuracy values per training epoch for both validation and training subsets.

2.3. Traditional Statistical Machine Learning Approaches



	benignware	hackware	malware
pixel265	10.07	11.52	12.81
pixel633	10.48	9.01	12.78
pixel1725	7.45	10.43	12.37
pixel1685	8.69	9.67	11.47
pixel1597	8.19	7.94	11.13
pixel1781	6.09	7.21	10.94
pixel1329	11.01	11.21	10.93
pixel1275	8.26	7.97	10.61
pixel1653	7.52	9.30	10.39
pixel1621	8.34	8.80	10.30

Figure 4. Most Determinant Byte (or Pixel) Positions for Firmware Class Assignment using Random Forest

To compare the effectiveness of deep learning, we solve the tabular equivalent of the firmware in pixel format but apply tree-based methods [33]. These methods such as decision trees and random forests offer robust interpretability for why they may assign a class label to the malicious firmware. The algorithmic choice between accuracy, speed, and simplicity thus provides additional model trade-offs and focuses future avenues for investigation. For example, particularly appealing output from tree-based methods includes the assignment of variable or feature importance in an automated way; among the 1024 bytes in the firmware's header, the method can extract the key positional bytes that signal a possible malicious operation.

3. RESULTS

3.1. Transfer Deep Learning

Table 1. Accuracy Results for Four Class MobileNetV2			
Class	**Lg. Accuracy (Test Samples)**	**Sm. Accuracy (Test Samples)**	**Microprocessor Scale (Test Samples)**
Benign firmware	0.98 (451)	0.98 (451)	0.99 (107)
Malware	1.00 (107)	0.94 (107)	0.86 (107)
Hack-ware	1.00 (16)	0.81 (16)	NA
Unknown	1.00 (101)	1.00 (101)	NA

Table 1 shows the accuracy for class determinations for the small (96x96) and large (224x224) images when custom training the MobileNetV2 architecture. The choice of small images (which are rescaled from the original 32x32) accommodates cameras for embedded systems such as Arduino BLE Sense micro-controllers. The accuracy for a class decision approaches 100% for the larger images and suggests the ELF headers provide a sufficiently rich pattern in the first 1024 bytes to assign a risk factor to each firmware binary. The training time versus accuracy (entropy loss) is shown in Figure 5. After 10 epochs, the network has effectively reached its plateau both for training and validation subsets. The execution times for this style of MobileNetV2 approaches real-time (equivalent to 30 frames per second), such that the overall processing for validating firmware might be limited only by the time to read the first 1024 bytes and flatten them to a decimal equivalent in pictures.

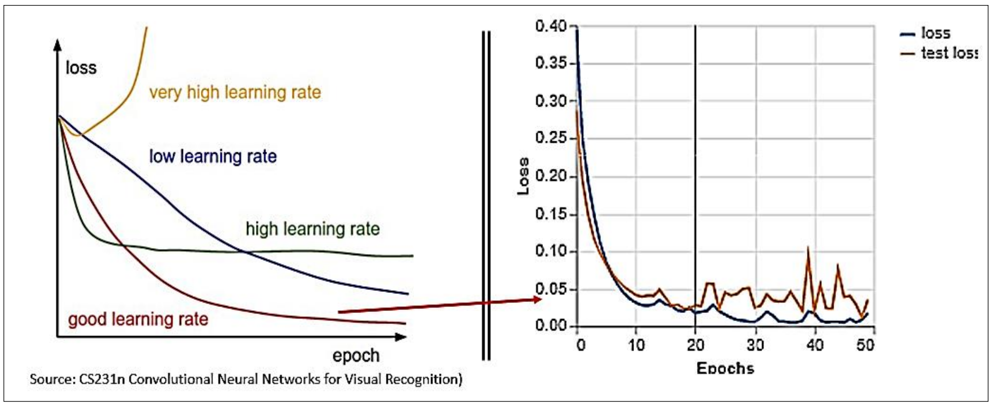


Figure 5. Learning Loss Rates over Time in Epochs (right). Characteristic Learning Curves shown left Highlight the Need to Select Efficient Learning Rates

To explore the effects of training sample size, we built smaller balanced models of 711 examples for binary classification of either malware or benign firmware. The two-class model shown in Table 1, Microprocessor Scale, showed an 86% accurate identification of malware compared to 99% for benign firmware. The resulting model, when uncompressed to a microprocessor occupies 8 kB (227 total lines) as an Arduino sketch in C++ (Arduino Nano 33 BLE board, with OV7670 camera module). The investigative intention centers on reading on the serial port of a low-power device and capturing the raw stream of grey-scale pixels representing firmware bytes.

3.2. Single Decision Tree

Figure 6 shows a single decision tree based on considering all 1024-pixel values but splitting firmware class determinations based on ranges of grayscale (or decimal-byte conversions) in the ELF header. Using a subset (4%) of the full training dataset and further holding out a 15% test dataset for evaluation, the decision tree method achieves 95.9% accuracy (4.1% error) across all three classes (benign firmware, hack-ware, and malware). This result is competitive with the deep learning approach (99+% accuracy, Table 1). Single trees offer the additional advantage of easier interpretability. One can, for instance, envision a simple algorithm for detecting malicious firmware by examining the decimal conversion of selected key binary bytes in the ELF header. Figure 6 also shows the most important 10 bytes as positions at 1285, 377, 298, and so forth. A shortcoming of this approach for single decision trees, however, stems from their brittleness, particularly when applied to test data outside of the narrow training threshold. If an attacker

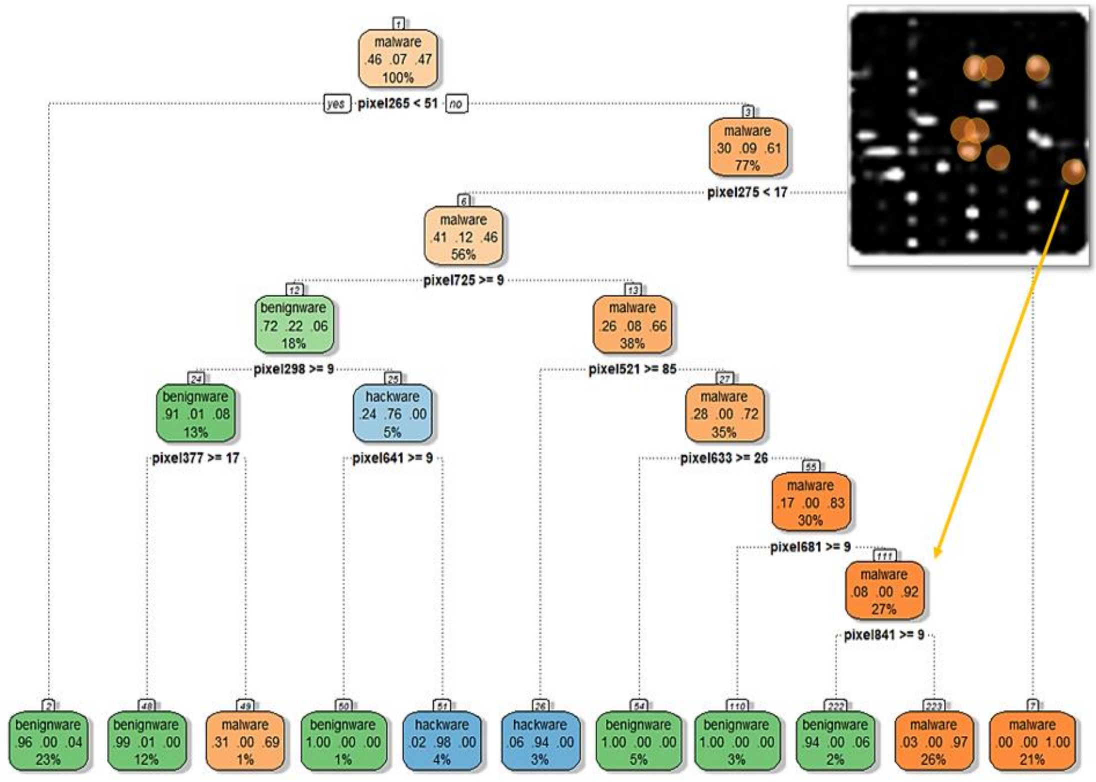

1 malware .46 .07 .47 100%
yes pixel265 < 51 no
3 malware .30 .09 .61 77%
pixel275 < 17
6 malware .41 .12 .46 56%
pixel725 >= 9
12 benignware .72 .22 .06 18%
13 malware .26 .08 .66 38%
pixel298 >= 9
pixel521 >= 85
24 benignware .91 .01 .08 13%
25 hackware .24 .76 .00 5%
27 malware .28 .00 .72 35%
pixel377 >= 17
pixel641 >= 9
pixel633 >= 26
55 malware .17 .00 .83 30%
pixel681 >= 9
111 malware .08 .00 .92 27%
pixel841 >= 9
2 benignware .96 .00 .04 23%
48 benignware .99 .01 .00 12%
49 malware .31 .00 .69 1%
50 benignware 1.00 .00 .00 1%
51 hackware .02 .98 .00 4%
26 hackware .06 .94 .00 3%
54 benignware 1.00 .00 .00 5%
110 benignware 1.00 .00 .00 3%
222 benignware .94 .00 .06 2%
223 malware .03 .00 .97 26%
7 malware .00 .00 1.00 21%

Figure 6. Single Decision Tree Applied to Firmware ELF Bytes

discovers the key 10 bytes for this method to assign a malware or hack-ware class to the binary, then the decision tree suffers from the same fragility as hash-based or signature methods. A single-byte change can render the detector ineffective. In the next section, we investigate whether ensembles of many decision trees can improve the generalizing capability of single trees.

3.3. Multiple Decision Trees, or Random Forest

To investigate the robustness of statistical methods compared to deep learning, Figure 7

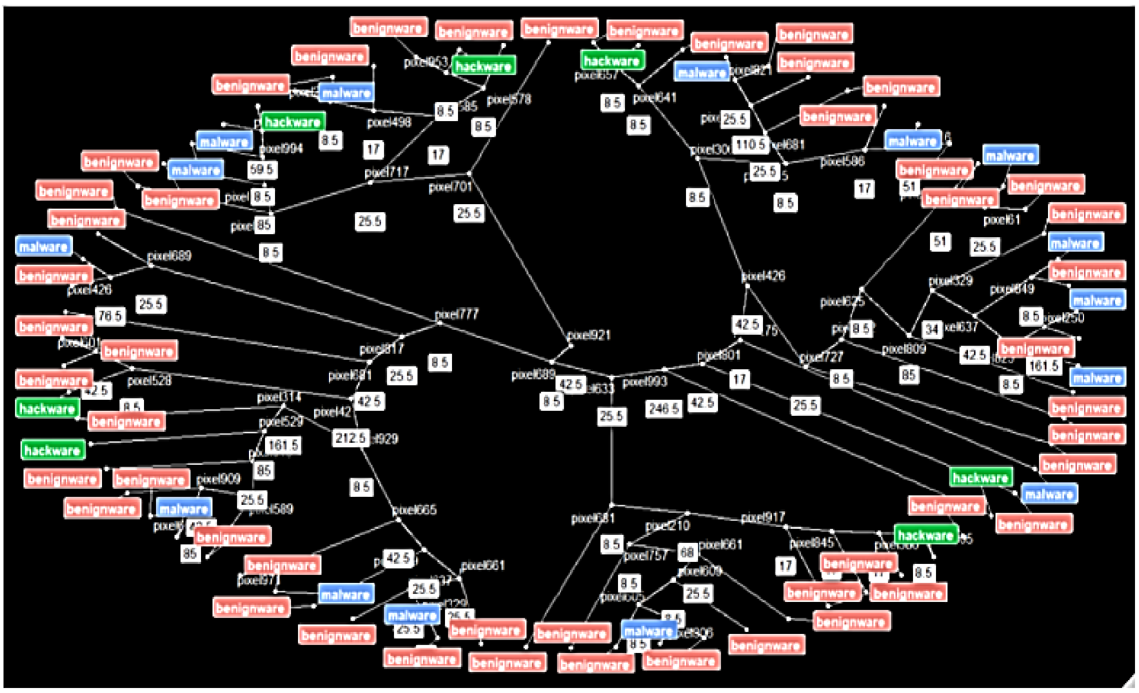


Figure 7. Random forest (tree 1) for firmware class

illustrates the application of a random forest [33]. The concept of combining 50-500 trees in a voting ensemble generates less brittle decisions. Compared to Table 1 for CNNs, the random forest achieves 100% class accuracy. The circular plot in Figure 7 is much denser with decision branches than the single tree shown in Figure 6. Starting in the center of the plot (origin), decision branches for (yes-no) choices span out until a labeled class can be identified by the outer (colored) tags. The resulting high accuracy model combines an ensemble of 500 such trees to render a perfect classification for withheld testing data. The approach of combining many (often weaker) learners to render an ensembled strong learner is well-known for its enhanced robustness and ability to generalize better than single trees when confronted with out-of-band or under-represented data. The results indicate that these statistical machine learning methods may offer alternatives for malware detection, particularly in cases where explainable AI represents a key concern. They further offer a path to generalizing image classification outside of the original training data, such as what cybersecurity assessments might call "zero-day" malware never before seen or classified in any training data.

4. Discussion and Conclusions

By applying deep (transfer) learning to converted images of firmware headers, an optimized neural network can classify malicious Executable and Linkable Files (ELF). The small (32x32) grayscale images match with a decimal conversion (0-15) of the raw binary and then are scaled to a wider (0-255) pixel value range. Each pixel represents a byte in order and the underlying pattern of malicious intent appears across the file and image nomenclature [22] for architecture, compiler, program name, etc. A procedure to under-sample the benign firmware better rebalances the dataset but leaves between 100-3000 images per class. This number of representative samples has previously been shown to be sufficient, particularly when not training the network from scratch but inherited the weighted features from a previous run on unrelated classes (transfer-learning) [32]. Future work can apply the large research efforts of MNIST derivatives to this firmware classification including simpler or more easily explainable algorithms that are tree-based methods. The research highlights an accurate tree-based method that offers additional interpretability advantages and suggests new ways to apply "if-then" filtering to ELF binaries before firmware updates. The image dataset is available on Kaggle [29] as both JPEG thumbnails and comma-separated value (CSV) files. The original dataset of ELF headers with labels by type is also available from GitHub [22].

ACKNOWLEDGMENTS

The authors would like to thank the PeopleTec Technical Fellows program for its encouragement and project assistance.

REFERENCES

[1] LeCun, Yann, Corinna Cortes, and C. J. Burges. "MNIST handwritten digit database." (2010): 18.http://yann.lecun.com/exdb/mnist/ and Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, 86(11):2278-2324, November 1998

[2] Cohen, Gregory, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. "EMNIST: Extending MNIST to handwritten letters." In *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2921-2926. IEEE, 2017.

[3] CV Online (accessed 01/2021) http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm

[4] Google Scholar search (accessed 01/2021), https://scholar.google.com/scholar?q=mnist and https://trends.google.com/trends/explore?date=all&q=mnist,ImageNet,%2Fg%2F11gfhw_78y

[5] Chen, Li, Song Wang, Wei Fan, Jun Sun, and Satoshi Naoi. "Beyond human recognition: A CNN-based framework for handwritten character recognition." In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pp. 695-699. IEEE, 2015.

[6] Image Classification on MNIST, (accessed 01/2021), https://paperswithcode.com/sota/image-classification-on-mnist

[7] Grim, Jirı, and Petr Somol. "A Statistical Review of the MNIST Benchmark Data Problem."http://library.utia.cas.cz/separaty/2018/RO/grim-0497831.pdf

[8] Schott, Lukas, Jonas Rauber, Matthias Bethge, and Wieland Brendel. "Towards the first adversarially robust neural network model on MNIST." *arXiv preprint arXiv:1805.09190* (2018).

[9] Cheng, Keyang, Rabia Tahir, LubambaKasangu Eric, and Maozhen Li. "An analysis of generative adversarial networks and variants for image synthesis on MNIST dataset." *Multimedia Tools and Applications* 79, no. 19 (2020): 13725-13752.

[10] Preda, Gabriel, Chinese MNIST: Chinese Numbers Handwritten Characters Images, (accessed 01/2021) https://www.kaggle.com/gpreda/chinese-mnist

[11] CoMNIST: Cyrillic-oriented MNIST, A Dataset of Latin and Cyrillic Letters, (accessed 01/2021) https://www.kaggle.com/gregvial/comnist

[12] Prabhu, Vinay Uday. "Kannada-MNIST: A new handwritten digits dataset for the Kannada language." *arXiv preprint arXiv:1908.01242* (2019).https://www.kaggle.com/higgstachyon/kannada-mnist

[13] Noever, David, Noever, Sam E.M. "Expressive Multimodal Integrated Learning (EMIL): A New Dataset for Multi-Sense Integration and Training", 2020 Southern Data Science Conference, August 12-14 2020, Atlanta, GA (poster) and Sign Language MNIST: Drop-In Replacement for MNIST for Hand Gesture Recognition Tasks, https://www.kaggle.com/datamunge/sign-language-mnist

[14] Mader, K Scott, Skin Cancer MNIST: HAM 10000, A Large Collection of Multi-Source Dermatoscopic Images of Pigmented Lesions, (accessed 01/2021) https://www.kaggle.com/kmader/skin-cancer-mnist-ham10000

[15] Xiao, Han, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms." *arXiv preprint arXiv:1708.07747* (2017).https://www.kaggle.com/zalando-research/fashionmnistSee also Fashion-

MNIST, (accessed 01/2021), https://paperswithcode.com/sota/image-classification-on-fashion-mnist and https://github.com/zalandoresearch/fashion-mnist

[16] Lu, Arlen, “Convert-own-data-to-MNIST-format” (accessed 01/2021) https://github.com/Arlen0615/Convert-own-data-to-MNIST-format

[17] Noever, D., & Noever, S. E. M. (2021). Overhead MNIST: A benchmark satellite dataset. arXiv preprint arXiv:2102.04266.https://www.kaggle.com/datamunge/overheadmnist and Github, https://github.com/reveondivad/ov-mnist

[18] Noever, D., & Noever, S. E. M. (2021). Virus-MNIST: A Benchmark Malware Dataset. arXiv preprint arXiv:2103.00602.

[19] Noever, D. A., & Noever, S. E. M. (2021). Image Classifiers for Network Intrusions. *arXiv preprint arXiv:2103.07765*.

[20] Freitas, S., Duggal, R., & Chau, D. H. (2021). MalNet: A Large-Scale Cybersecurity Image Database of Malicious Software. arXiv preprint arXiv:2102.01072

[21] Chen, L., Sahita, R., Parikh, J., Marino, M. (2020). STAMINA: Scalable Deep Learning Approach for Malware Classification, https://www.intel.com/content/dam/www/public/us/en/ai/documents/stamina-scalable-deep-learning-whitepaper.pdf

[22] Partush, N. (2021). Labeled-Elfs, https://github.com/nimrodpar/Labeled-Elfs

[23] Kairajärvi, S., Costin, A., &Hämäläinen, T. (2019). Towards usable automated detection of CPUarchitecture and endianness for arbitrary binary files and object code sequences. arXiv preprint arXiv:1908.05459.

[24] Clemens, J. (2015). Automatic classification of object code using machine learning. *Digital Investigation*, *14*, S156-S162.

[25] Xie, H., Abdullah, A., &Sulaiman, R. (2013). Byte frequency analysis descriptor with spatial information for file fragment classification. In *Proceeding of the International Conference on Artificial Intelligence in Computer Science and ICT (AICS 2013)*.

[26] Constantin, L. (2015). Cisco warns customers about attacks installing rogue firmware on networking gear, Network World. Aug 10, 2015. https://www.networkworld.com/article/2970954/cisco-warns-customers-about-attacks-installing-rogue-firmware-on-networking-gear.html

[27] Microsoft Threat Intelligence Center(2019). Corporate IoT – a path to intrusion. https://msrc-blog.microsoft.com/2019/08/05/corporate-iot-a-path-to-intrusion/

[28] Greenberg, A. (2019). The Untold Story of the 2018 Olympics Cyberattack, the Most Deceptive Hack in History. Wired Magazine. https://www.wired.com/story/untold-story-2018-olympics-destroyer-cyberattack/

[29] Noever, D. (2021). IoT Firmware Image Classifier: Rendered ELF Binaries by Class as Malware, Kaggle. https://www.kaggle.com/datamunge/iot-firmware-image-classification

[30] Sandler, Mark, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510-4520. 2018.

[31] Salehi, Sohail. *ImageMagick Tricks*. Packt publishing ltd, 2006.

[32] Warden, P. "How many images do you need to train a neural network?" (2017).https://petewarden.com/2017/12/14/how-many-images-do-you-need-to-train-a-neural-network/

[33] Morales-Molina, C. D., Santamaria-Guerrero, D., Sanchez-Perez, G., Perez-Meana, H., & Hernandez-Suarez, A. (2018, November). Methodology for malware classification using a random forest classifier. In 2018 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC) (pp. 1-6). IEEE.