Analyzing CNN Model Performance Sensitivity to the Ordering of Non-Natural Data

Randy Klepetko* and Ram Krishnan†
Department of Electrical and Computer Engineering
University of Texas at San Antonio, San Antonio, Texas, USA
Email: *randy.klepetko@my.utsa.edu, †ram.krishnan@utsa.edu

***Abstract*—Convolutional Neural Networks (CNN) have had significant success in identifying and classifying image datasets. CNNs have also been used effectively in classifying non-visual datasets such as malware and gene expression. In all of these applications, CNNs require data to be organized in a certain order. In the case of images, this order is naturally presented. However, in the case of non-visual data, this order is sometimes not naturally defined and hence requires an artificially defined order. The sensitivity of a CNN model's performance to various artificial orders of non-natural datasets is not well-understood. In this paper, we investigate this problem by experimenting with various orders of a dataset derived from malware behavior in a cloud auto-scaling environment. We show that the ordering can have a major impact on the performance of the CNN and offer some insights on how to derive one or more orderings that could provide better performance.**

***Index Terms*—Convolutional Neural Networks; Security; Malware Detection; Cloud IaaS; Deep Learning;**

I. Introduction

Over the past decade with the advances in CNN systems, performance of machines to classify images has surpassed the ability of humans [1]. It has been shown that CNN are effective in classifying sequences of text [2], acoustical samples [3], and genomic data [4]. In all of these cases, the data is ordered as rows and columns in the form of a matrix[1]. The ordering of the rows and columns is naturally defined, such as the spatial location of a pixel in an image, the temporal position of a character/words in a sequence of text and that of waveforms in audio samples, or physicality of amino acids found in a strand of DNA. But what about data that does not have a naturally defined order? For instance, in [5], Abdelsalem et al. show that CNN can also be used in dynamically identifying malware using computer process metrics as the data source. In [6], Smith et al. compare several machine learning algorithms, including CNN, in dynamically detecting malware by examining the stream of commands called by the malware process. Lihao and Yanni ,in [7], use CNN in the application of production level quality control in automobile tire manufacturing, using various production level measurements they make during the manufacturing process. Unlike images, audio signals and DNA, in these applications of CNN, there is no naturally occurring order that could be leveraged to order the rows and columns in the matrix. When we examine data such as images that has natural relationships and compare points between rows and columns, we can find mathematical relationships between adjacent pixels. If we run a statistical analysis on the values between pixels, we will find that pixels of the same object will be highly correlated, while pixels of different objects or edges will have a low correlation. Golinko et al. in [8], uses a similar approach for ordering generic data sets when using a CNN as a feature extractor for other classifying algorithms.

In this paper, we investigate this problem by experimenting with various row and column orderings of a dataset derived from malware behavior in a cloud auto-scaling environment. We show that the ordering can have a major impact on the performance of the CNN and offer some insights on how to derive one or more orderings that could provide better performance. We utilize the dataset provided by Abdelsalam et al. [5] for this research. In their work they arrange the data in matrices by rows, each identified to belong to a specific computer process, and columns, each associated with a sampled machine performance metric. Unlike data sources that are derived from nature, there is no naturally defined relationship that exists between these rows and columns. It is the goal of this research to determine if variations to the ordering of non-natural data has an affect on the performance of a CNN in detecting malware from machine metrics and in particular can an optimum ordering be determined by using relationships similar to those found in the natural world.

The contributions of this paper are:

- Show that ordering of rows and columns has a major impact on the performance of CNN.
- Validate that maintaining ordering between data samples and sources is imperative to achieve an optimum CNN performance, improving from 92% to 98% accuracy.
- Show that statistical correlation of samples is a strong predictor of a good performing order, with an additional performance improvement from 98% to 99%.

The remainder of the paper is organized as follows: Section II discusses related work using CNN with non-natural data. Section III outlines the methodology including a description of the ordering of the data and the CNN model. Section IV describes the analysis results. Section V

[1]In reality, this would be an n-dimensional vector. However, we use the term matrix interchangeably with that of a vector of 2 dimensions.

978-1-7281-0875-9/19/$31.00

TABLE I: Virtual machines performance metrics

Metric Category	Description
Status	Process status, Current working directory
CPU information	CPU usage, CPU user space, CPU system/kernel space, CPU children user space, CPU children system space.
Context switches	Voluntary context switches, Involuntary context switches
IO counters	Read requests, Write requests, Read bytes, Write bytes, Read chars, Write chars
Memory information	Swap memory, Proportional set size (PSS), Resident set size (RSS), Unique set size (USS), Virtual memory size (VMS), Dirty pages, Physical memory, Text resident set (TRS), Library memory, Shared memory
Threads	Used threads
File descriptors	Opened file descriptors
Network information	Received bytes, Sent bytes
Group Information	Group ID real, Group ID saved, Group ID effective

summarizes and concludes this paper.

II. Related Work

In this section we present other research that examines using non-natural data as the source for the CNN analysis. CNN were initially designed as a mimic to human vision and since have achieved super human performance in this task, but what about data sources that do not relate to vision?

In [7], Lihao and Yanni analyze the quality of rubber tire treads based on the parameters measured during the manufacturing process. There are four levels to the manufacturing process with eleven metrics sampled at each level. They vectorize these all of these parameters, with some filtering for noise, and then feed those vectors into a CNN. They were able to achieve a 94% accuracy with this process. Other than noise filtering they did not discuss data preparation or how it was organized as it was fed the CNN.

Golinko et al. in [8], examine using a one dimensional CNN as a feature extractor for other machine learning algorithms (kNN with k=1, SVN, and RF) with non-natural "Generic" data and examine if ordering of the source data for the CNN has any impact on performance of the final classifying algorithm. They propose using statistical correlation as a method for identifying relationships of adjacent data and show that not pre-ordering the data for CNN feature extraction is detrimental to performance. They show ordering by correlation offers significant improvement in most cases, especially for kNN and SVN, improving final average accuracy from 76% with no feature extraction to 82% if the features were ordered by correlation prior to CNN feature extraction.

One field that CNN has been used on non-natural data is in the dynamic analysis of malware. This is where malware is intentionally injected into a machine and the resulting logs and reports are studied for unique patterns and results that could be used in identifying future infections.

Smith et al. in [6], perform dynamic malware analysis using process calls made by the executed malware code as the data source. Executed code submits a series of commands calls in sequence. These calls are command strings (ie. "ssh") which are pre-processed via a one hot encoding and commands that are issued during the same time segment are included into a single on hot encoded vector. A series of these vectors represent the executed code as the calls are submitted to the kernel over time. These vectors are then analyzed as a group by several different machine learning techniques, comparing malware executed code and non-malware executed code. They show that a CNN can have a 94% accuracy with a 95% precision and 89% recall. They do not discuss the ordering of the one hot vector.

In [9], Tobiyama et al. compile more information regarding the process command calls from the machine logs. These include time of the process, name of the process, process ID, name of the command, the current working directory, the result from the command, and any other information included when the command was called. They then feed this information into a pre-trained RNN for feature extraction, taking the vector output of the RNN and feed that into a CNN for analysis. With this process they were able to achieve an AUC of 96%. They do not discuss the ordering of the one hot vector prior to feeding it to the network for analysis.

Abdelsalem et al. in [5], use metrics retrieved by hypervisors in a cloud environment that track the individual processes on a virtual machine. This is a set of 35 metrics that are captured for each process running on the VM. They are then compiled into a process row, metric column matrix which is supplied to a 2D-CNN. They achieved an 89% accuracy, but did not attempt to optimize the data by maintaining process row ordering or identify a preferred metric column ordering during pre-processing. Our research expands on the techniques discussed in this paper.

Based on this related work, our research goals are:

- Identifying a preferred or even an optimum order for any data that is supplied to a CNN for analysis.
- Improving dynamic malware detection when using a CNN by properly pre-processing the data with regards to row and column ordering.

III. Methodology

Our goal is to determine if modification of the ordering of rows and columns of the data matrices affect performance of the deep learning model. So first we examine the data.

A. Overview of the malware dataset

The data we are using are metrics sampled by process, as found in Table I. A sample is taken every ten seconds from

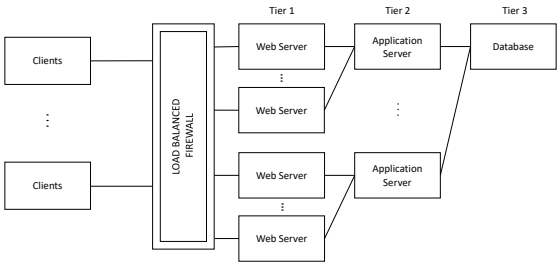


Fig. 1: 3-Tier Web Service

several Linux virtual machines. These virtual machines are configured in a typical scalable three tier web service environment, diagrammed in Figure 1. One is a database server, while others are web servers supplying html pages to clients or Wordpress™application servers supplying packaged data between the database and web servers. Due to the scalability of the cloud environment, there could be multiple application or web servers.

While the samples are taken from all of the machines in the environment over thirty minutes, we inject a malware executable into an application server at the fifteen minute mark, continuing to take samples for the remaining portion of the collection experiment. From this application server, we have a set of data labeled "clean", before the malware was injected, and "infected", after the injection.

After 114 different experiment's each on a separate environment and with a unique malware, we end up with samples from 912 virtual machines. 114 of the application servers were infected which resulted in over 29 million process samples, 2.9 million from the infected machines.

B. Organization of the malware dataset

These metrics are then organized into a "process matrix" where every row of the matrix represents a process p_i, and every column of the image represent a particular metric m_i.

$$\mathbf{X}_t = \begin{bmatrix} & m_1 & m_2 & \dots & m_k \\ p_1 & \vdots & \vdots & \dots & \vdots \\ p_2 & \vdots & \vdots & \dots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_n & \vdots & \vdots & \dots & \vdots \end{bmatrix}$$

C. Metric Column Ordering

Since we have a "process matrix" a question remains on how to order the rows and columns? When we examine visual images, we are able to identify objects and edges. If we run analysis on the pixel values of an image we will find that pixel values are statistically correlated, or the value of a pixel is closely related to the value of an adjacent pixel within the same object. By converse if we examine the relationship of pixels on edges and between objects, we'll find there is little, or even a negative correlation. We use this knowledge as a basis for ordering.

Statistical correlation is the association found between parameters, causal or not. This means that change in one parameter corresponds to a related change in the other parameter. In statistics this is mathematically defined by:

$$\rho_{m_i m_j} = \frac{E(m_i m_j) - E(m_i)E(m_j)}{\sqrt{E(m_i^2) - E(m_i)^2} \cdot \sqrt{E(m_j^2) - E(m_j)^2}} \quad (1)$$

where $E(m_i)$ is defined as the expected value or mean of m_i.

In MySQL™we ran statistical analysis on all possible pairs of numeric metrics for all 29 million samples. This resulted in a table of correlation values, a real number from -1 to +1, for every pair of metrics. We then organized several ordered lists for each of the metrics using the correlation value as a parameter for constructing the order.

The first order we label "Correlated" where adjacent metrics have correlation values that are maximized:

$$\underset{\rho}{\operatorname{argmax}}(\rho_{m_i m_{i+1}}) \forall i = 1...k \quad (2)$$

or we define an order that maximizes the following equation:

$$\sum_{i=1}^{k} (\rho_{m_i m_{i+1}}) \quad (3)$$

This order is designed to generate "objects" from related metrics for the CNN identification.

The second order we label "ABS-Correlated" where adjacent metrics have the absolute value of the correlation maximized:

$$\underset{\rho}{\operatorname{argmax}} \left|\rho_{m_i m_{i+1}}\right| \forall i = 1...k \quad (4)$$

or the order maximizes the following equation:

$$\sum_{i=1}^{k} \left|\rho_{m_i m_{i+1}}\right| \quad (5)$$

This takes in consideration that strongly negative correlated pixels could represent different parts of the same object.

As a counterpoint, we also attempt to generate a poorly performing order, we label "Anti-Correlated". This is where the correlation values are minimized:

$$\underset{\rho}{\operatorname{argmin}} \left|\rho_{m_i m_{i+1}}\right| \forall i = 1...k \quad (6)$$

or the order will minimize the absolute value equation:

$$\sum_{i=1}^{k} \left|\rho_{m_i m_{i+1}}\right| \quad (5)$$

For comparison, we randomly derive ten additional columns orderings. This give us the following list of column ordering experiments:

- Correlated

- ABS-Correlated
- Anti-Correlated
- Ten Random

Most of the data metrics sampled are numerical values which are normalized and included in the ordering scheme mention above, but we also had two string metrics, "status" and "cwd" or current working directory. Correlation has no bearing on string values, so these metrics needed to be handled separately. In the paper [5] by Abdelsalem et al. the "status" was one hot coded and included as additional columns added to the front of the matrix while "cwd" was ignored. This increased the number of columns to 45. We decided to include the "cwd" as a metric in this set of analysis. The string was one hot encoded into a set of vectors and added between the status one hot vectors and the ordered metrics. This expanded the number of columns from 45 to 75.

D. Process Row Ordering

When examining Abdelsalem's et al. data pre-processor from [5], we discovered that the row ordering of the "process matrix" was set per VM, but not across all VM's. Different VM's ran different processes, many unique per VM, and the number of all of the processes could be large and computationally expensive if all included. It was obvious that before we decide on an arbitrary order to test, we needed to construct a method that establishes an order that remains systematic across all experiments.

In the analysis performed by Abdelsalem et al. in [5], they limited the number of process rows to 120, organized by name. Most VM's ran around 105 processes, but some ran more than 120.

When we ran data queries on the processes, by name, we discovered that 129 of them were executed on more than one machine. We call these the "non-unique" processes, and decide that these will make the first 129 rows of our image. If a sample was selected, and one of these "non-unique" processes was not running, we supplied that row with zero values.

At the bottom of the matrix is where we place any "unique" process. We left 21 place holders for "unique" processes, which are normally zeroed out if none are present. In cases where there were more than one "unique" process they were ordered alphanumerically in this section. As a result the row count increases to 150.

We attempted to identify statistical correlations between processes, but that query proved to be infeasible with questionable results. Instead we decided to study several different methodologies for ordering the 129 non-unique rows using relationships that already exists between the processes.

One of the easiest methods is to simply allow the process name dictate the row order. This row ordering we label "Alphanumeric". This approach takes the perspective that related processes my have related names.

Another approach is to identify referential relationships between processes. As one process initiates the execution of another process, the first process is referred to as the parent and the initiated process is referred to as a child. Our data samples include the parent and children relationships so we use this information to identify all of the children and gather them together in a group. We can determine which of these children are initiated more often, so we use this as the ordering among the children. We label this row ordering "Sibling Relationships".

We also are able to identify the number of virtual machines that execute a process, and the number times that a process is executed for all of the experiments. We call the number of machines a VM count, and number of executions as the PID count. We use these parameters to establish a couple of other ordering.

"VM/PID Count" uses the VM count descending as the order for the processes, and when there is tie for the number of machines that call two different processes, we use PID count, descending, as the tie breaker. In this ordering the first row is the process that is run on the most number of machines and is call most by those machine, while the last row is the process called the least.

"PID/VM Count" uses the total number of calls, descending, as the primary parameter that determine the row ordering, and in cases where two processes are called equal number of times for all of the experiments, the number of machines making those calls breaks the tie.

For comparison, we randomly derive ten additional row orderings. This give us the following list of row ordering experiments:

- Alphanumeric
- Sibling Relationships
- VM and PID Count
- PID and VM Count
- Ten Random

Between ten randomly generate rows and ten columns, this gives us 100 different arbitrarily devised matrix orderings with which we compare performance to our derived orderings.

IV. Evaluation

A. Testbed

We run our pre-processing and analysis using a desktop with the following specifications:

- Central Processor Unit: Intel©Core™i7-8700 CPU @ 3.2 GHz x 12
- Memory: 15.6 GB
- Graphical Processor Unit: GeForce™GTX 1070i/PCIe/SSE2
- OS: 64-bit Ubuntu©18.04LTS (Gnome 3.28.2)
- CUDA™: 10.0
- Python: 3.6

We used Tensorflow™with Tensorboard™as the underlying engine to perform the CNN analysis. We found that preprocessing to produce the all of the analyzed data for a single matrix ordering definition from the raw data contained

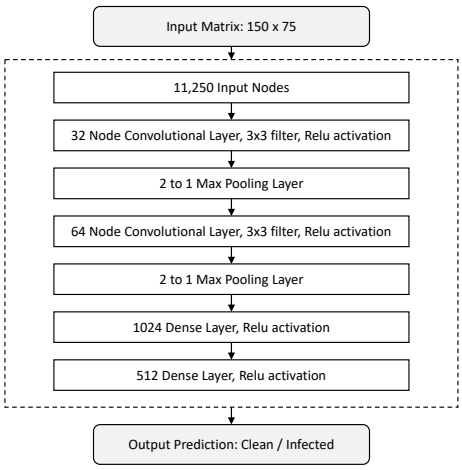


Fig. 2: LeNet-5 CNN Model

within a MySQL™database would take about two hours but did not require the GPU. The CNN analysis on a single matrix ordering definition leveraged the GPU and would finish in about 2.5 hours. When analyzing several matrix orderings at a time, the pre-processing would not interfere with the GPU operations so both could be daisy chained on a single machine for improved throughput.

B. CNN Model

The CNN matches the model that was used by Abdelsalem et al. [5], a LENET-5 version. We duplicated it to compare results. Figure 2 shows the layout of the CNN model. Several details:

- Two Convolutional layers where the first consisting of 32 nodes and the second consisting of 64 nodes. Each convolutional layer uses a 3x3 filter and relu as the activation function.
- After each convolutional layer is a max-pooling layer with a downsize of a factor of two to one.
- Two dense layers, the first consisting of 1024 nodes and the second consisting 512 nodes. Relu was also used as the activation function for the dense layers.
- Predictive layer uses binary cross entropy loss function to produces a probability that the data sample examined is infected or not.
- We ran training for 20 epochs with a batch size of 64.

C. Results

Our analysis splits the infected machines samples into three groups: 60% for training, 20% for validation and 20% for testing. All of the data for a single virtual machine was included in one of these groups, or in other words, no machine data was split between groups.

To properly study the affects of column and row ordering we first built a background of 100 randomly ordered matrices for comparison. Since malware infections are rare compared to the normal machine activity we decided that it is more important to compare the precision/recall curves. In figure 3 and 4 we see a sample spread in performance for 100 (10x10) random variations of the row and column ordering of the source data matrix versus other row and column ordering that we methodically defined.

We compare this result to the paper [5] by Abdelsalem et al. where row ordering between experiments was not taken into consideration in figure 3a. It is obvious that ordering the rows between experiments is imperative to reach high performing results.

Now we compare our 100-random curves with different metric column orderings. For each methodical metric column ordering we used all of the various process row orders so the column order can be analyzed with some independence from row ordering. In figure 3b we show correlated metric columns with various row ordering. Correlated columns appear to reside in the upper spectrum of the randomly ordered curves.

When we look at the absolute value of the statistical correlation between metrics we find the following graphs 3c. In this set of curves we not only find that they reside the upper spectrum of the random curves, but possibly an optimum result appears, one better than any random ordering.

Next we examine our counter example, Anti-Correlated columns ordering. As expected, in 3d we find that the curves do not align to the upper end on the randomly ordered curve spectrum, and most curves show rather poor results in comparison, residing on the lower end of the spectrum.

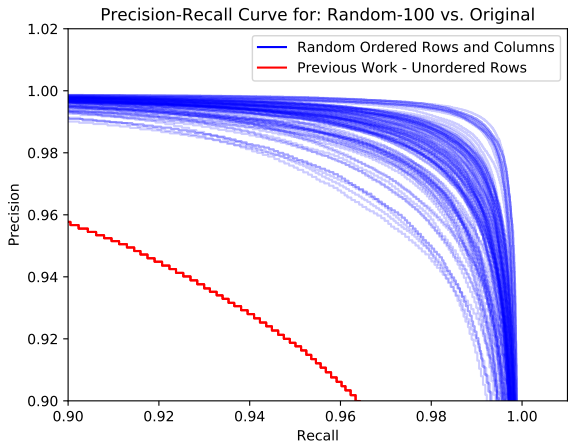
Next we examine our results for the defined row orderings. Again in each methodical row order case we used all of the various column orderings for some analysis independence between the rows and columns.

The results for the orderings based off of initial alphanumeric relationships is found in graph 4a. We see in that alphanumeric row ordering results are spread relatively evenly among the randomly ordered curves.

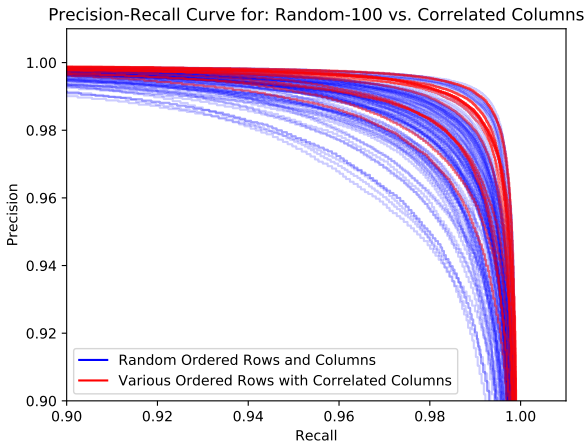
Examining the results for the orderings that are based off of sibling relationships in graph 4b we see our results are spread among the random curves. In general they reside on the higher than average end of the randomly ordered curve spectrum, but nothing stands out as exemplary.

Studying the results for the orderings that are based off of the number of machines that make the process call followed by the number of execution or Process ID counts in graph 4c. we see in our results are spread among the random curves. Again they are higher than average, but nothing stands out as exemplary.

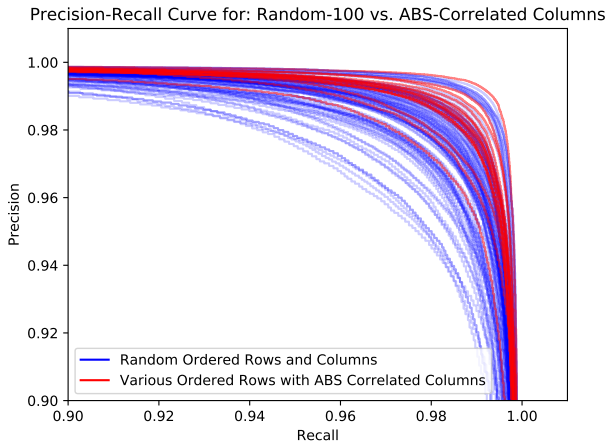
Studying the results for the orderings that are based off of Process ID counts followed by number of machines that call the process in graph 4d. we see in our results are again spread among the random curves. These curves though



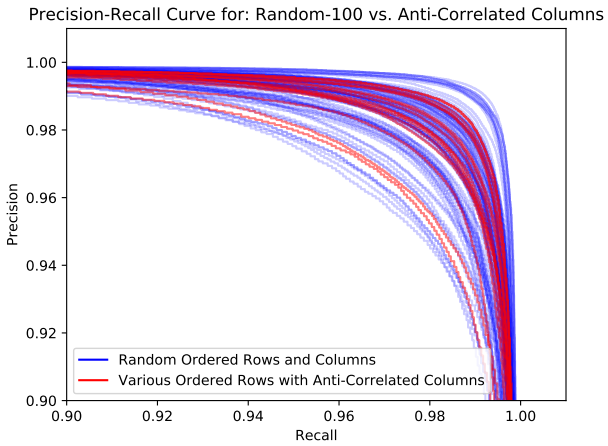
(a) Vs. Non-Ordered Rows between VM Samples



(b) Vs. Correlated Columns



(c) Vs. Absolute Value Correlated Columns



(d) Vs. Anti-Correlated Columns

Fig. 3: 100 Randomly Ordered Rows and Columns PR curves vs. Different Columns Ordering Sets

reside on the lower set of random curves, informing us this is not a desired ordering.

As a comparison for other studies that only take the accuracy in consideration for evaluation, we include the following two graphs that display the means and standard deviation spread for the various row and column ordering performance accuracy. In 5b we share the accuracy spread for the various row ordering and in 5a we share the accuracy found different column ordering including the non-ordering from the previous research by Abdelsalam et al. [5].

V. Conclusion

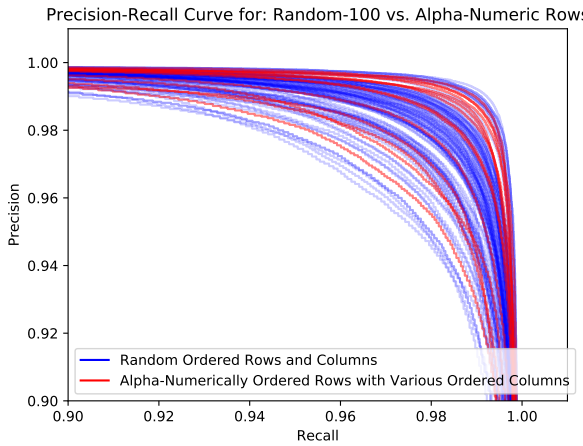
A. Summary

From this set of experiments it is clear that when using CNN analysis on non-natural data, especially process metric data for malware classification, some care is required in organizing source data to achieve improved results. It also shows that using some form of statistical correlation between rows and columns may produce an optimum result. It includes that malware detection using process metrics as the data source for CNN can achieve a high degree of accuracy ($> 99\%$) as long as the ordering of the rows and columns of the data matrix are properly defined.

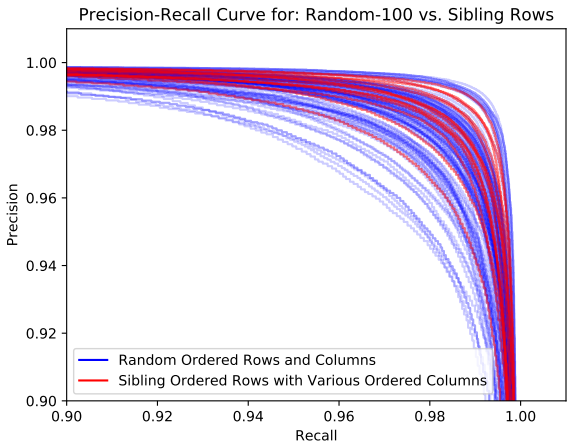
B. Further Work

Based on these results, and those found in other papers, there is additional work that can be performed to further this research.

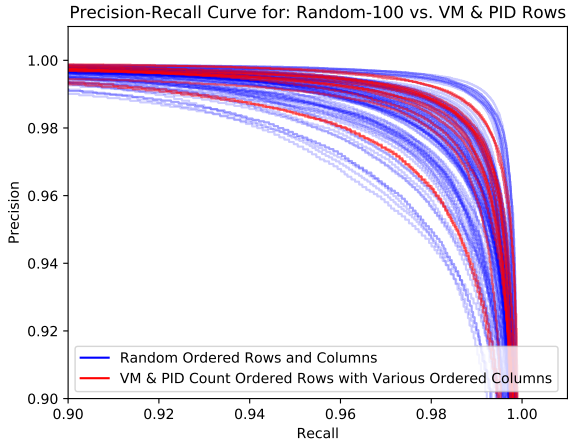
- With this data set we were able to identify relationships between metric columns through statistical correlation that provided preferred results, but the process row ordering results were moderate at best. Some work could be performed to identify relationships between process rows that could produce optimal results.
- Currently the data pool consists of only 114 Linux malware samples. An increase the in data samples, including non-Linux (Windows^{TM}) malware, would give this methodology confidence in tackling security issues in many environments.



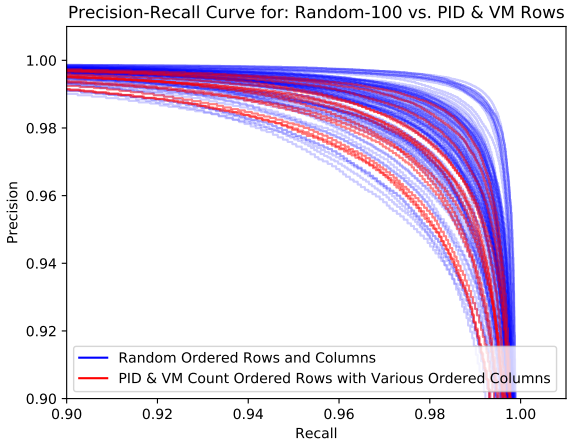
(a) Vs. Alphanumeric Rows



(b) Vs. Sibling Related Rows

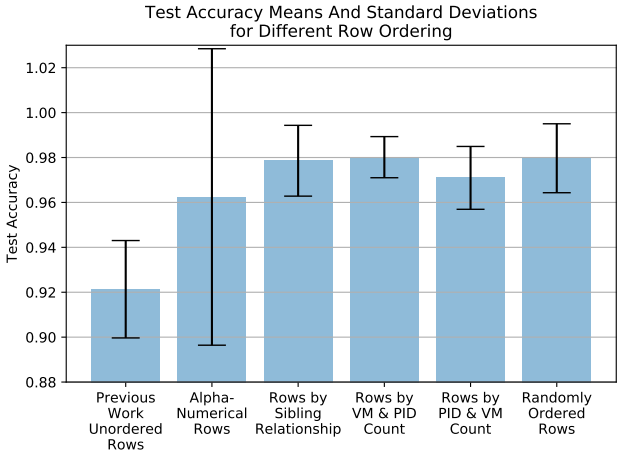


(c) Vs. Rows Ordered by VM count then PID count

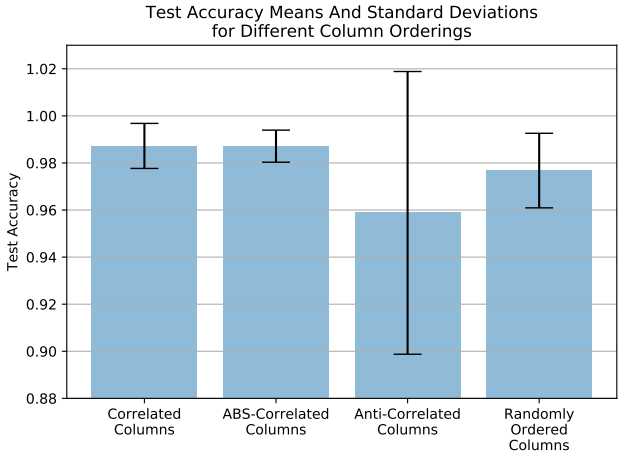


(d) Vs. Rows Ordered by PID count then VM count

Fig. 4: 100 Randomly Ordered Rows and Columns PR curves vs. Different Row Ordering Sets



(a) Accuracy of different row ordering



(b) Accuracy of different column ordering

Fig. 5: Test Accuracy Mean and Standard Deviation for Different Row and Columns ordering

- We limited the deep learning model to a simple CNN Lenet-5 model. Other models could provide better results. Time based models such as LSTM have potential with this data source. Also using the compiled data sets against a model ambivalent deep learning algorithms such as Auto-Keras™could be an option.
- We limited our analysis to data derived by malware on a cloud environment. This methodology could be potentially valuable in classification problems in other environments. Identifying new classification questions using novel non-natural data sources, perhaps industrial, for testing this process is another avenue for future research.

ACKNOWLEDGMENT

This work is partially supported by NSF Grants HRD-1736209 and CNS-1553696, NSA CAE-CO Initiatives, and DoD ARL Grant W911NF-15-1-0518.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[2] J. Y. Lee and F. Dernoncourt, "Sequential short-text classification with recurrent and convolutional neural networks," *CoRR*, vol. abs/1603.03827, 2016. [Online]. Available: http://arxiv.org/abs/1603.03827

[3] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: an overview," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8599–8603.

[4] P. Mobadersany, S. Yousefi, M. Amgad, D. A. Gutman, J. S. Barnholtz-Sloan, J. E. Velázquez Vega, D. J. Brat, and L. A. D. Cooper, "Predicting cancer outcomes from histology and genomics using convolutional networks," *Proceedings of the National Academy of Sciences*, vol. 115, no. 13, pp. E2970–E2979, 2018. [Online]. Available: https://www.pnas.org/content/115/13/E2970

[5] M. Abdelsalem, R. Krishnan, Y. Huang, and R. Sandu, "Malware detection in cloud infrastructure using convolutional neural networks," *IEE 11th International Conference on Cloud Computing*, 2018.

[6] M. Smith, J. Ingram, C. Lamb, T. Draelos, J. Doak, J. Aimone, and C. James, "Dynamic analysis of executables to detect and characterize malware," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 16–22.

[7] W. Lihao and D. Yanni, "A fault diagnosis method of tread production line based on convolutional neural network," in *2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS)*, Nov 2018, pp. 987–990.

[8] E. Golinko, T. Sonderman, and X. Zhu, "Learning convolutional neural networks from ordered features of generic data," in *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Dec 2018, pp. 897–900.

[9] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, "Malware detection with deep neural network using process behavior," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, June 2016, pp. 577–582.