Analyzing CNN Models' 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, TX, USA
randy.klepetko@my.utsa.edu, ram.krishnan@utsa.edu

Abstract. Convolutional Neural Networks (CNN) have revolutionized image recognition technology, and has found uses in various non-image related fields. When dealing with non-natural data, where the ordering of various parts of a data sample is not dictated by nature, it is known that a model trained on certain orderings of the data performs better than models trained on other orderings. Understanding how to best order the training data for improving CNN performance is not well-studied. In this paper, we investigate this problem by examining several different CNN models. We define a functional algorithm for ordering, show that order importance in CNNs is model dependent and that depending on the model, statistical relationships are an important tool in establishing order with better performance.

Keywords: Convolutional Neural Networks · Data preparation · Security · Malware detection · Cloud IaaS · Deep learning

1 Introduction

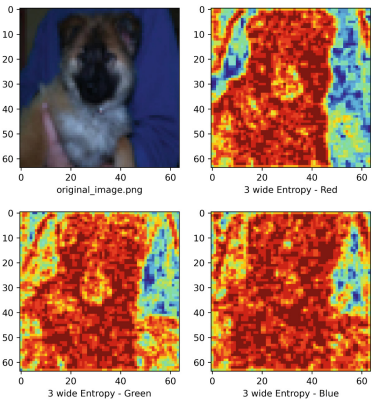
Recent explosion in CNN architectures has pushed computer image recognition [8] to an art form. It has provides a variety of options depending on the need [5]. They are also used in non-image related fields, so understanding how they work with images should help us leverage their use in these other areas.

It has been shown that entropy can be used to both increase detail [24] and reduce noise [2]. By examining the entropy of an image, for example, the dog in Fig. 1a, and comparing the activation values found by analyzing it with a shallow CNN, Fig. 1b, we can see it identifying patterns in entropy. We hypothesize that these new CNN models are finding novel ways of making identifiable information out of these patterns of entropy.

Exploration has been made in using CNN in fields other than image classification. Text [14], sound samples [4], and medical diagnostics of DNA [20] are examples of how this technology has other uses. Oftentimes these sources of data have a naturally defined order such as the acoustical waves in a sound or DNA in

This work is partially supported by NSF grants HRD-1736209 and CNS-1553696.

R. Krishnan et al. (Eds.): SKM 2021, CCIS 1549, pp. 130–148, 2022.
https://doi.org/10.1007/978-3-030-97532-6_8



(a) Image with a 3 wide Entropy of Primary Colors

(b) CNN Level-2 Activations of Image

Fig. 1. Image processed by CNN

a sequence. But many times these data sources do not have a naturally defined order, for example, a series of sensors on an automated vehicle [22]. In most "*non-natural*" cases, the researcher defaults the matrix order to a structural relationship between features usually established by an arbitrary specification. We use the term "*non-natural*" as a definition of ordering sources that were not defined in nature. This is opposed to "*unnatural*" which leads to the idea that they were ordered by something super-natural.

Our previous research showed that if high accuracy and precision are desired, using a non-natural structural order is not preferred when training a shallow convolution neural network model. We found that using statistical relationships as a basis for order does improve performance. We hypothesize that this holds true for the other styles of CNN architectures.

A particular subset of non-natural data that has gained interest is in detecting security issues. For example, raw IP traffic [16,23], computer process metrics [1], and industrial sensors [10] are examples where researchers are evaluating the use of CNN in security-related fields. The ability for a CNN to examine a large number of features from which extract the important subsets is what make CNN successful. Properly compiling various sources of data in a structure for a deep learning algorithm to analyze should be of concern when using CNNs.

Security can have many forms of data, all from a single source, for example, computer metrics [1]. Some are integers, others floats, include strings, all with various ranges. In a different scenario, a researcher could include audio, video, and or bio-metric packages to augment and enrich the source. We hypothesize that how the data is structured and prepared is imperative to use CNN successfully.

The contributions of this paper are:

- Show that ordering of rows and columns has a major impact on the performance of CNN, but how much is model dependent.
- Define a methodology for ordering the data by statistical relationships.
- Show that using statistical relationships to define matrix order is a strong predictor of a good performing order, but the exact statistical relationship can depend on the model of CNN.
- Increase the state of malware detection technology by providing data preparation tools that improve CNN performance when analyzing security data.

The remainder of the paper is organized as follows: Sect. 2 discusses related work using CNN with non-natural data. Section 3 outlines the methodology including, a description of ordering the data. Section 4 describes the analysis procedure and evaluation results. Section 5 summarizes and concludes this paper.

2 Related Work

2.1 Convolutional Neural Networks and Non-natural Data

As we understand the CNN capabilities, its use cases continue to expand. In this section, we examine the use of CNN by other researchers using non-naturally ordered data sets.

Lihao and Yanni analyze the quality of rubber tire treads in [15] using the parameters measured during the manufacturing process. With four levels to the procedure and eleven metrics sampled at each level, this provided a 4×11 matrix. After vectorizing these parameters, filtering for noise, they then feed them to a CNN, achieving a 94% accuracy. The order of the grid construction wasn't discussed.

Using a one dimensional CNN as a feature extractor for other machine learning algorithms (k-Nearest Neighbor with $k = 1$, Support Vector Machine, and Random Forest), Golinko et al. in [6], examine with non-natural "Generic" data if the ordering of the source data for the CNN has a performance impact on the final classifying algorithm. Using statistical correlation as a method for identifying relationships of adjacent data they show that not pre-ordering the data for CNN feature extraction is detrimental. They show using correlation as an ordering offers improvement in most cases, especially for kNN and SVN, improving accuracy from 76% with no feature extraction to 82% if the features were ordered by correlation prior to CNN feature extraction.

In [22] Park, et al. used information from robotic sensors and actuators to design a robot collision detection system using 66 features. They tested both a Support Vector Machine Regression and a one-dimensional CNN and were able to show that the CNN would perform better if it trained with enough data, but the SVMR performed better with less training. The construction of vector order wasn't discussed.

With connected and automated vehicles, Van Wyk, et al. [22] used cross-related sensor data (local speed, GPS location, and accelerometer) fed through

an analyzer to identify whenever any of the sensors behaved anomalously. They tested different analyzers using a Kalman Filter, CNN, and a CNN-KF hybrid. Each had its unique benefits. Order of the grid constructing wasn't discussed, but trivial with three sensors over time.

2.2 Convolutional Neural Networks and Security

In security-based applications, CNNs have found value. Their ability to extract features out of a large data pool enables the algorithm's non-linear space to find patterns instead of statically looking for distinct signatures, allowing the dynamic/online detection of zero-day attacks. These data sources are often non-natural.

After minor pre-processing of raw IP traffic packets which included stripping the physical protocol layer, Zhang et al. [23], then they analyzed the resulting grids using CNN, LSTM, and a hybrid of the two, for both binary classification (benign/maleficent) and multi-classification (benign + 10 maleficent types). He shows they all achieve quite remarkable, near-perfect results. Differences being for binary classification the hybrid is slightly better than a CNN, which is better than LSTM. With multi-classification, CNN may have some minor advantage in precision over the hybrid, but LSTM is behind both. Data order was defined per the packet specification by the order of packets received.

Using process metrics as they are reported from hypervisors in a cloud environment Abdelsalem et al. in [1], places them in a grid-like structure looking for malware as it is injected into the virtual machines. Per time segment this produces a set of 35 metrics that are captured for each process running on the VM. They are compiled into a process row metric column matrix, which is supplied to a Lenet-5 [17] CNN. Using the order as found in the logs and specifications, they achieved an 89% accuracy. Using the same data set and ordering scheme, McDole et al. [18] follow up with research analyzing different CNN architectures. With ResNet [7] and DenseNet [11] he showed that Dense-121 performed the best at 92%, but Lenet-5 trained in an order of magnitude less time and detected in one-third. Kimmell et al. [13] includes the use of other deep learning models, Recurrent Neural Networks, by testing the validity using Long Short Term Memories and Bi-Direction LSTMs. In it, they explore if the order has an effect on training and discover that it does affect performance metrics for both models. For example, a precision of 99.95% with one random order and 98.46% with another.

Our previous research expands on the techniques discussed by Abdelsalem et al. [1] by exploring the relationship between ordering of the rows, columns, and CNN performance. We identify several structural relationships on which to base our ordering scheme, we include the use of a statistical relationship as an option for ordering the metric columns, and we compare those against a background of random orderings. We found that by establishing order using statistical correlation as a basis, we increased overall performance and achieved a 99% accuracy in detecting injected malware. We also show that using structural relationships as an ordering appears to have no more advantage than a random

Table 1. Virtual machine process 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

order, but statistical relationships offer some insight. Based on this related work, our research goals are:

- Further explore finding a preferred or even an optimum order for any data that is supplied to a specific CNN model for analysis.
- Improve dynamic malware detection by choosing the proper CNN model and pre-processing the data with regards to row and column ordering.
- Explore the use of this data set with later models of CNN architecture.

3 Methodology

3.1 Dataset - Metric by Process Grids

The source of the data are samples taken from virtual machines in a cloud IaaS environment. These virtual machines are arrayed as a LAMP stack hosted web-site. The application server is injected with malware halfway through the experiment. Each sample is for a specific process running on the VM kernel and contains a series of M number of metrics per process (Table 1) during a segment in time. Stacking P number of processes that are captured during a single slice of time results in the matrix:

$$\mathbf{X}_t = \begin{bmatrix} & m_1 & m_2 & \dots & m_M \\ p_1 & x_{m_1 p_1} & x_{m_2 p_1} & \dots & x_{m_M p_1} \\ p_2 & x_{m_1 p_2} & x_{m_2 p_2} & \dots & x_{m_M p_2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_P & x_{m_1 p_P} & x_{m_2 p_P} & \dots & x_{m_M p_P} \end{bmatrix}$$

For our experiments, the 35 metrics expanded through one hot encoding to $M = 75$ metric columns and we made available room in the matrix for as many as $P <= 150$ process rows. The 29+ million process samples were organized around 114 experiments (infections), and consisted of 31,064 matrices, about half of which are considered infected. The experiments were split between 80% training, 20% validation, and 20% testing. The entire grid set for each experiment was included in the group it was assigned, so no experiment was split between training, validation, and testing.

3.2 Row and Column Ordering Algorithms

This paper is to demonstrate if row/column ordering effects performance of different CNN models. Our initial method was to randomly sort the rows and columns. We choose ten rows and column orders which combined give us 100 unique ordering to use as a backdrop for comparison.

In our previous work we explored the use of structural relationships as one method for establishing an order. We found several relationships as determined by specification, log location, process number, parent/child and sibling status, related virtual machines, and naming convention. In these cases we found on average they performed no better than the random option if not worse. Since these ordering methods were previously defined we include them in our processing and as part of the general backdrop along with the random 100. We do not examine them specifically in the evaluation section of this paper.

Perhaps images provide us some insight on how to best order our matrices. CNN's are used to identify objects. What makes up an object in an image? Statistically, an object is a set of highly related pixels. All of the pixels will have a similar shade. Pixels outside the object boundaries usually have few patterns that match inside an object. This edge can be found using the statistical correlation relationship minimum. It is this fact that led to many advances in image compression techniques [12,17,21].

Table 2. Metric and process correlation functions

Metric statistical correlation function	
$\rho_{m_i m_j} = \frac{E(x_{m_i} x_{m_j}) - E(x_{m_i})E(x_{m_j})}{\sqrt{E(x_{m_i}^2) - E(x_{m_i})^2} \cdot \sqrt{E(x_{m_j}^2) - E(x_{m_j})^2}}$	(1)
Process statistical correlation function	
$\rho_{m_k p_i p_j} = \frac{E(x_{m_k p_i} x_{m_k p_j}) - E(x_{m_k p_i})E(x_{m_k p_j})}{\sqrt{E(x_{m_k p_i}^2) - E(x_{m_k p_i})^2} \cdot \sqrt{E(x_{m_k p_j}^2) - E(x_{m_k p_j})^2}}$	(2)

We hypothesize that we should create *artificial objects* by grouping the rows and columns to increase the average statistical relationship between neighboring features while decreasing the overall entropy of the image. In our previous paper, we found a relationship, statistical correlation $\rho_{m_i m_j}$ as shown in Table 2,

between metric columns m_i and m_j for all processes, comparing results from a LENET-5 style CNN with *relu*, we found that ordering based on statistical correlation improved performance. We attempted to disperse the *artificial objects* by minimizing the correlation between columns and it had a negative impact on performance. We include these column orderings in our evaluation details using other CNN models. This consists of three relationship functions, metric correlation (Table 2), the absolute value of the correlation $\rho_{ABSm_im_j} = |\rho_{m_im_j}|$ to increase object edge creation, and anti-correlation, $\rho_{ANTIm_im_j} = 1 - |\rho_{m_im_j}|$, to test a counter hypothesis dispersing the objects and increase the entropy.

In our previous work, we struggled to derive a statistical relationship for the process rows. Since there could be as many as 150 processes statistically related over the 35 metrics, each sample unique per process, our initial queries became infeasible. They suffered from a *vanishing correlation* when a large set of samples that are not related to the feature are included in the calculations. For this research, we pared down the queries so only related a pair of processes p_i and p_j over a single metric m_k were calculated at a time. We reduced the data set for this specific relation value to only include samples when these two processes were running on the same machine at the same time. This reduced the query time from what was months, to all process pairs around a single metric, $\rho_{m_kp_ip_j} \forall i, j$, in roughly 24 h. We then incremented through each metric. Once these calculations were finished, we had a full set of process pair correlation values per metric, $\rho_{m_kp_ip_j} \forall i, j, k$.

Summing the correlations for a single pair we had a statistical relationship value between the processes $\rho_{SUMp_ip_j}$:

$$\rho_{SUMp_ip_j} = \sum_{k=1}^{M} (\rho_{m_kp_ip_j}) \tag{3}$$

Since we processed the row relationship values per metric before we summed them, we purposely chose which order of metric to derive these relationship values. We already had a relative importance order in our metric correlations from our previous research (Eq. 1 above). By summing all of the columns correlations for a single metric:

$$\rho_{TOTm_i} = \sum_{j=1}^{M} (\rho_{m_im_j}) \tag{4}$$

This is the *total metric correlation* on which to order their importance, largest to smallest. We also do the same for process rows, resulting in *total process correlation*:

$$\rho_{TOTp_i} = \sum_{j=1}^{P} (\rho_{SUMp_ip_j}) \tag{5}$$

Along with our fully correlated rows ordered derived from Eq. 3, we took the opportunity to tests some other options derived from this function. Like metric columns, we test similar relationship ideas with both the absolute values of

the correlations, $\rho_{ABSp_ip_j} = \sum_{j=1}^{M} \left|\rho_{m_kp_ip_j}\right|$ and anti-correlations $\rho_{ANTIp_ip_j} = \sum_{j=1}^{M}(1 - \left|\rho_{m_kp_ip_j}\right|)$.

With this statistical relationship value, we rank the importance of each metric column and process row with each other. We built a methodology to construct the order. The process is generic and modular with regards to the data source ,f_i row or column, and the function used to derive the statistical relationship value $\rho_{f_if_j}$. The ordering methodology uses the steps in Algorithm 1.

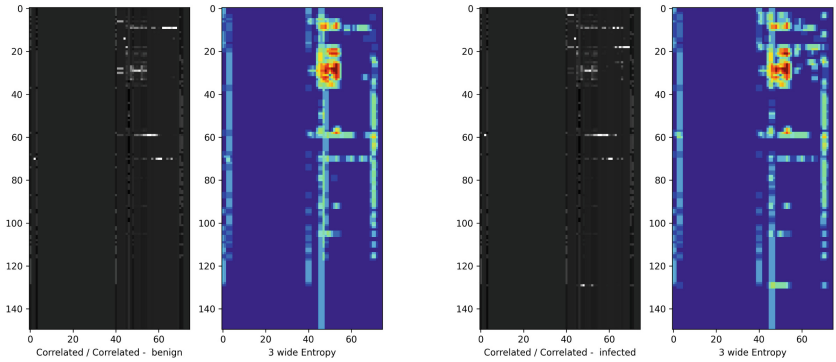
Algorithm 1: Derive Statistical Relationship Order

For features along an axis, f_i, define a function, $\rho_{f_if_j} \forall i, j$;
From $\rho_{f_if_j}$ define $\rho_{TOTf_i} \forall i$;
Create a selection pool of features $P \ni f_i$;
while $P \neq \varnothing$ **do**
  Create an empty bidirectional queue Q for features f_i;
  Find $max(\rho_{TOTf_i}) \forall f_i \in P$;
  Place the corresponding feature $f_{max(\rho)}$ onto Q;
  Remove $f_{max(\rho)}$ from P;
  Create two pointers left, L, and right, R; $L, R \in Q$;
  Point L and R towards $f_{max(\rho)}$ in Q;
  while $P \neq \varnothing$ *and not(**STOP**)* **do**
    if $\exists \rho_{f_Lf_i} \forall f_i \in P$ *or* $\exists \rho_{f_Rf_i} \forall f_i \in P$ **then**
      Find $max(\rho_{f_Lf_i}, \rho_{f_Rf_i}) \forall f_i \in P$;
      Place the feature $f_{max(\rho)}$ next to f_L or f_R on Q;
      Remove $f_{max(\rho)}$ from P;
      Move the pointer, L or R, to the new feature $f_{max(\rho)}$ in Q;
    else
      Stack current queue Q into a final ordered axis V;
      STOP
    end
  end
end
Result: A vector V of features f_i that are ordered by the relationship function, $\rho_{f_if_j}$
Derive Statistical Relationship Order

Occasionally, there are ties. This was especially true for the anti-correlated function. Many pairs of processes rows had no correlation between them. We would settle ties by examining the next set of neighbors to see which set increased the relative total relationship value of the entire grid.

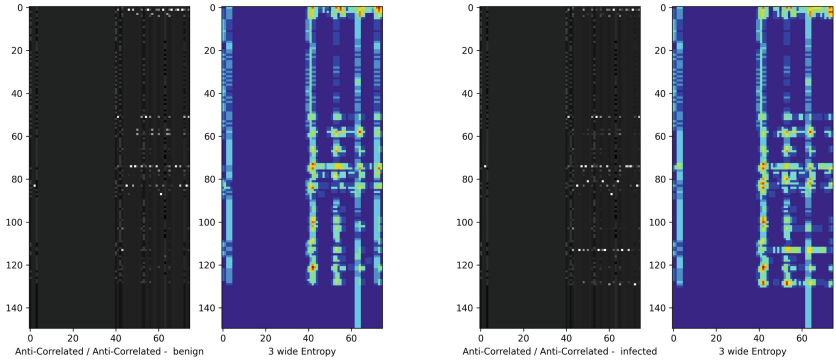
After compiling the statistically related orders with the previously defined order sets, we have a total of 252 distinct grid orders to compare. A visual example of the grids in different ordering sets is shown in Fig. 2 and Fig. 3.

We show two slices, one benign and another infected, using different row and column ordering schemes. It includes a 3-square pixel entropy filter plot to highlight possible patterns the CNN may be detecting. One order set, Fig. 2, has both rows and columns correlated while the other, Fig. 3, has them anti-correlated. You can see how we construct objects using the correlated order while dispersing them into tiny objects using the anti-correlated order.



(a) Benign (b) Infected

Fig. 2. Visual plot of correlated samples with 3 wide entropy



(a) Benign (b) Infected

Fig. 3. Visual plot of anti-correlated samples with 3 wide entropy

4 Evaluation

4.1 Test Beds

We run our pre-processing and analysis using two desktops with the following specifications:

Desktop-1

- Central Processor Unit: Intel©Core™i7-8700 CPU @ 3.2 GHz × 12
- Memory: 15.6 GB
- Graphical Processor Unit: GeForce™GTX 1070i/PCIe/SSE2
- OS: 64-bit Ubuntu©20.04.2 LTS (Gnome 3.36.8)
- CUDA™: 11.1
- Python: 3.6

Desktop-2

- Central Processor Unit: Intel©Core™i7-9700K CPU @ 3.6 GHz × 8
- Memory: 15.5 GB
- Graphical Processor Unit: NVIDIA GK210GL (Tesla K80)
- OS: 64-bit Ubuntu©20.10 LTS (Gnome 3.38.3)
- CUDA™: 11.2
- Python: 3.6

We used Tensorflow™v2 with Tensorboard™, the underlying engine, to perform the CNN analysis. Comparing between these machines, we found that the Tesla could handle larger CNN models with two cores and more GPU memory, while the GeForce machine would process about 30% faster with the later CUDA capable features.

4.2 CNN Models - Chosen Through Experimentation

Our previous research examined the use of a shallow CNN model, Lenet-5 with *relu* as an activation function. In this research, we wanted to see if our statistical relationship hypothesis would hold with other forms of CNN. We initially experimented with Resnet-50 and found that the training times took longer per epoch and more epochs than Lenet-5. Lenet-5 would usually saturate training in 20 epochs, but Resnet-50 would take as long as 50. We shifted to ©Auto-Keras and by 20 epochs it would settle on a plain CNN with a couple of dense layers but fail to produce any meaningful performance.

We then took a modularly broad but targeted approach by re-coding our test ground to use the recently released ©Keras application set of deep learning models. Using a limited set of ordered experiments, we test model training saturation. Because of our methodology, using the same data set for the different models was simply changing the model name within the script. Our post calculation analysis found that five models would saturate training much quicker than the others, within three epochs, so we chose to compare those in order of their release date:

– Inception-V3 [19].
– ResNet-18 [7].
– Xception [3].
– MobileNet [9].
– DenseNet121 [11].

To help in our analysis, we examined the model summary so we could identify the parameters count and see if there might be some relationship between that and order performance via the architecture design. These details are found in Table 3.

Table 3. Model parameter count and process times

CNN architecture	Parameter count		Desktop-2 median 3-epoch train time (min)
	Functional layers	Dense layers	
Inception-V3	21,802,208	12,290	2:45
ResNet-18	11,186,698	162	11:43
Xception	20,860,904	61,442	6:03
MobileNet	3,230,338	6	1:20
DenseNet-121	7,031,232	16,386	3:54

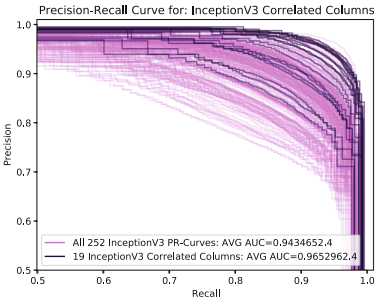
4.3 Result Plots

Since malware infections are rare compared to normal machine activity, we decided to compare the precision/recall curves. We start by showing the results for the Inception V3 model. In Fig. 4, we see all of the PR curves as the light background with the dark lines representing a subset of PR curves that are generated running the model over a particular order set. Note that these plots are scaled in to 50%–100%. Here we see Inception prefers correlated columns and ABS-correlated rows, while and correlated rows offer another well performing alternative, but anti-correlated rows should be avoided.

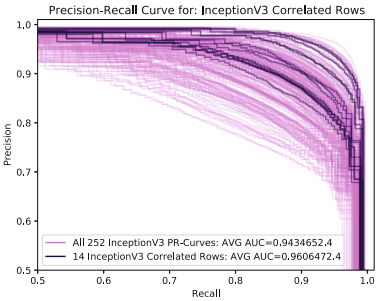
We follow with the results from ResNet-18 in Fig. 5. Note that these plots are at 0–100% scale. It's obvious by the wide varieties in PR curves that this model is very susceptible to minor changes in order. For this model anti-correlated rows and columns perform better than average, while the other orderings have only minor variation around the poor average.

Our next model is Xception, and the results are found in Fig. 6. Note that this model seems order ambivalent with near perfect results every time, but we see that the statistically related order performs well if not better than average. Only the ABS-correlated columns fell below average, but this was by only 0.0007 AUC. It appears the best performance is found using correlated rows and anti-correlated columns.

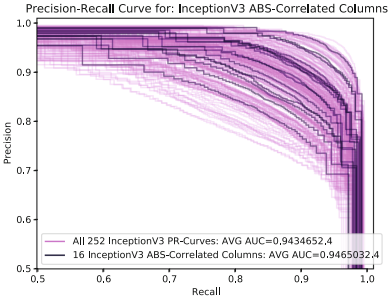
We included MobileNet as a small format option with it's intention to be used in mobile devices. You can find the results in Fig. 7. Like ResNet-18, MobileNet



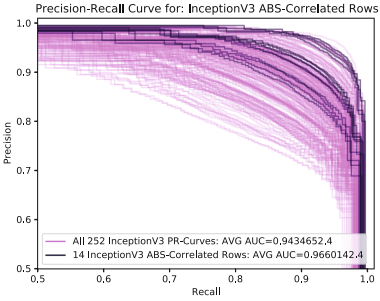
(a) Correlated Columns



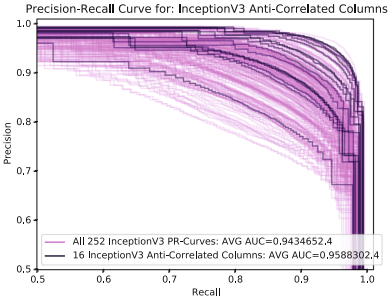
(b) Correlated Rows



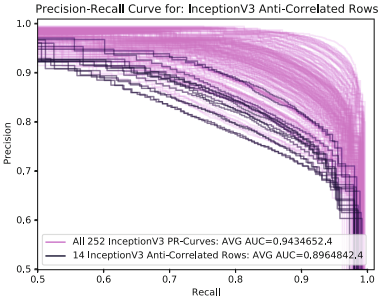
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows

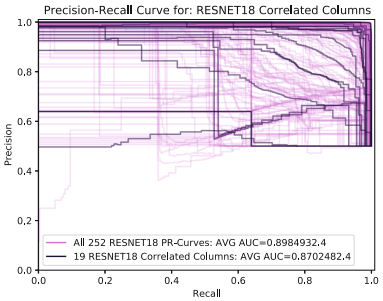


(e) Anti-Correlated Columns

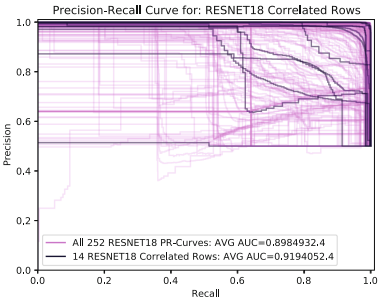


(f) Anti-Correlated Rows

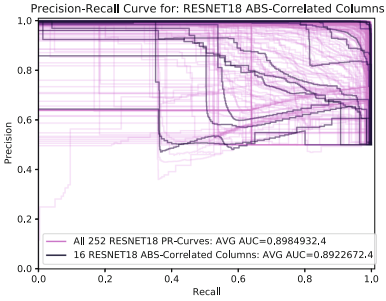
Fig. 4. Inception V3 CNN model PR curves



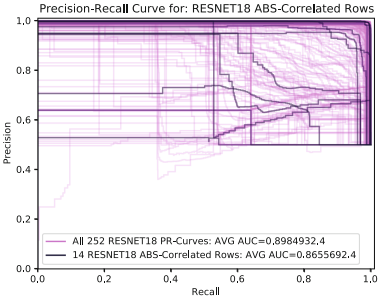
(a) Correlated Columns



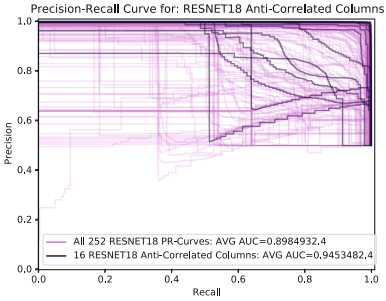
(b) Correlated Rows



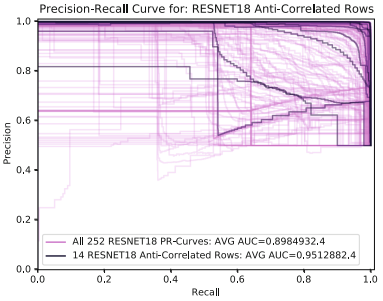
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows

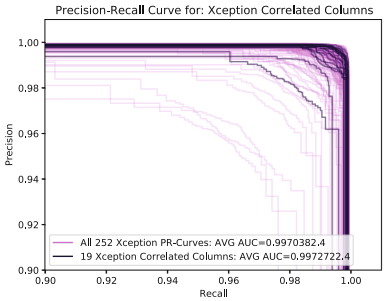


(e) Anti-Correlated Columns

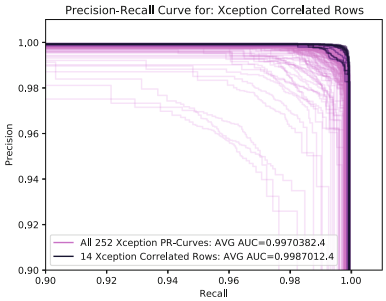


(f) Anti-Correlated Rows

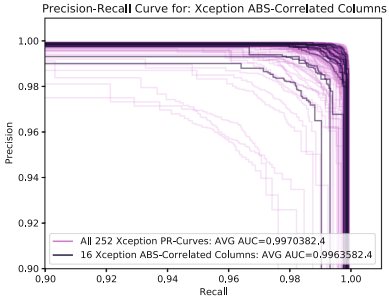
Fig. 5. ResNet-18 CNN model PR curves



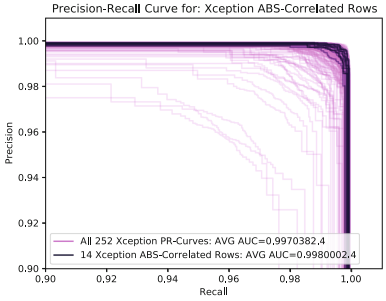
(a) Correlated Columns



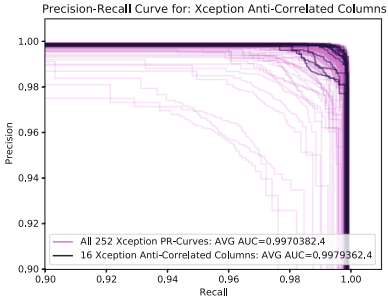
(b) Correlated Rows



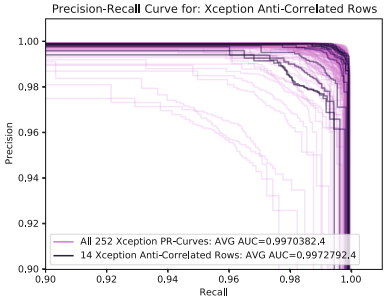
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows

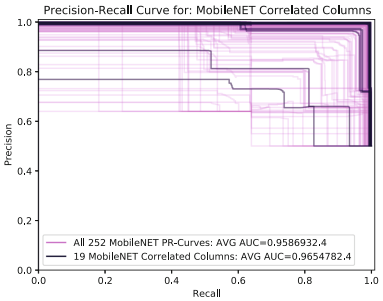


(e) Anti-Correlated Columns

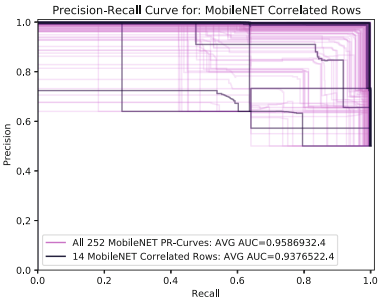


(f) Anti-Correlated Rows

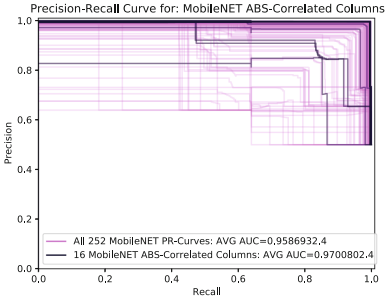
Fig. 6. Xception CNN model PR curves



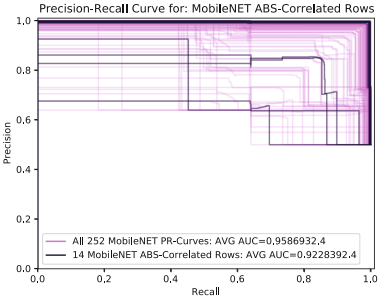
(a) Correlated Columns



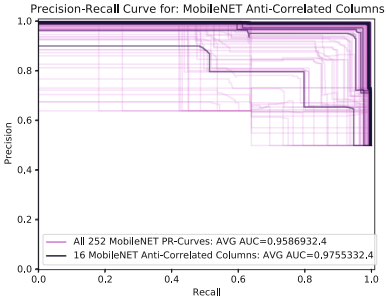
(b) Correlated Rows



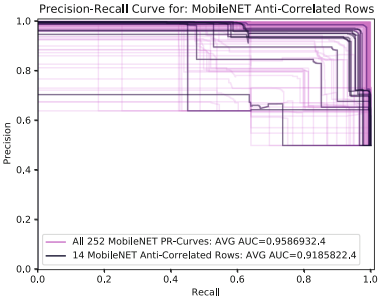
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows

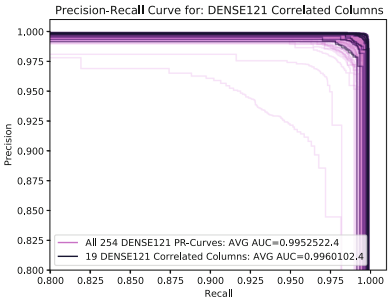


(e) Anti-Correlated Columns

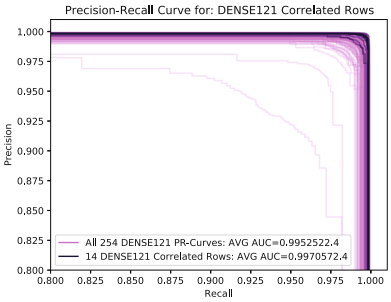


(f) Anti-Correlated Rows

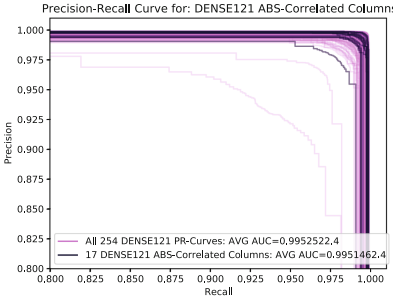
Fig. 7. MobileNet CNN model PR curves



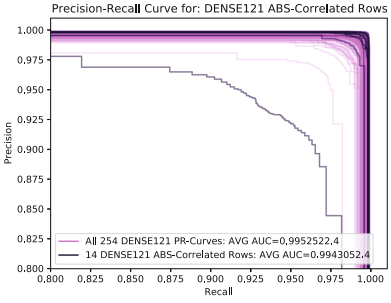
(a) Correlated Columns



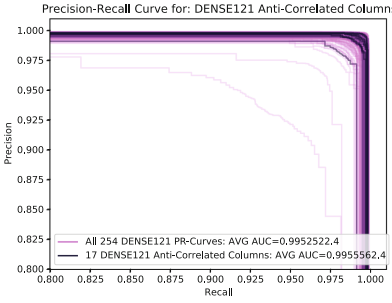
(b) Correlated Rows



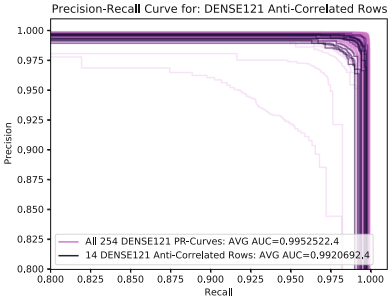
(c) ABS-Correlated Columns



(d) ABS-Correlated Rows



(e) Anti-Correlated Columns



(f) Anti-Correlated Rows

Fig. 8. DENSE121 CNN model PR curves

seems to be very reactive when there are changes in the order. We have these plots at full zoom, 0–100%, to observe all of the curves. Unlike ResNet-18 (0.898 AUC), it appears to respond better on average (0.958 AUC). It also appears that it loves any statistical relationship in column order, but choosing a random order is better than anything we analyze for row order.

In our final examination, we analyze Dense-121 found in Fig. 8. This like Xception had a vary high AUC regardless of row or column order, with almost near-perfect results every attempt. Only a couple of curves drop below 97%, and we had the figures zoomed in at 80%–100% for that purpose. You can see that correlated rows and columns are the best option, but all of the statistical relationships seem to provide an average if not better result.

Looking back at the parameter count Table 3, we see that those architectures with fewer parameters in the final dense decision layers were fragile with response to order and performance. Even slightly changing the grid order in these models has great impact on the results. It appears the opposite is true, that the more parameters in the dense decision layers reduce the impact of changing the order.

We see in almost every model that using a statistical relationship to determine a proper order does improve performance, but identifying which relationship to use requires some experimentation. We see that ResNet architectures find granularity in the detail with an anti-correlated order, while most of the remaining models prefer using regular correlation. MobileNet is the only model that wasn't responsive, and that was only when ordering the rows. It responded very well when using any statistical relationship to order the columns.

5 Conclusion

This research gives us several points for our hypothesis.

- Order can have a major impact on CNN performance, especially when few neurons are in the final dense decision layer.
- Statistical correlation is a solid benchmark for good performing order for most CNN, but not guaranteed, especially for models with small dense layers.
- Resnet architectures prefer the anti-correlated ordering.
- It appears that MobileNet order response behavior is axis independent.
- Xception proved best with this data set and performed well using correlation as an ordering scheme.

This leads us to several open questions:

- Do these observations hold true for other data?
- Does anti-correlation observation hold true for deeper versions of Resnet?
- Why does MobileNet respond so differently when comparing rows and columns? Is it the axis size difference?
- Can we leverage our understanding of entropy to further improve CNN performance?

It's these questions that lead us into our next topic.

5.1 Future Work

To further our understanding on how order has an affect on CNN performance, we plan on continuing our research by:

- Examine the use of this technique using other security data sets, the CIC-IDS-2017 in particular.
- See if this technique holds true for non-security related data sets, especially in industrial and medical fields.
- Identify if there are other statistical relationships that could improve the performance of the CNN using data preparation alone.

References

1. Abdelsalem, M., Krishnan, R., Huang, Y., Sandu, R.: Malware detection in cloud infrastructure using convolutional neural networks. In: IEEE 11th International Conference on Cloud Computing (2018)
2. Avula, S.B., Badri, S.J., Reddy P, G.: A novel forest fire detection system using fuzzy entropy optimized thresholding and STN-based CNN. In: 2020 International Conference on COMmunication Systems NETworkS (COMSNETS), pp. 750–755 (2020). https://doi.org/10.1109/COMSNETS48256.2020.9027347
3. Chollet, F.: Xception: Deep learning with depthwise separable convolutions (2017)
4. Deng, L., Hinton, G., Kingsbury, B.: 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, pp. 8599–8603 May (2013). https://doi.org/10.1109/ICASSP.2013.6639344
5. Elhassouny, A., Smarandache, F.: Trends in deep convolutional neural networks architectures: a review. In: 2019 International Conference of Computer Science and Renewable Energies (ICCSRE), pp. 1–8 (2019). https://doi.org/10.1109/ICCSRE.2019.8807741
6. Golinko, E., Sonderman, T., Zhu, X.: Learning convolutional neural networks from ordered features of generic data. In: 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), pp. 897–900, December 2018. https://doi.org/10.1109/ICMLA.2018.00145
7. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR abs/1512.03385 http://arxiv.org/abs/1512.03385 (2015)
8. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: The IEEE International Conference on Computer Vision (ICCV), December 2015
9. Howard, A.G., et al.: Mobilenets: efficient convolutional neural networks for mobile vision applications (2017)
10. Hu, Y., Zhang, D., Cao, G., Pan, Q.: Network data analysis and anomaly detection using CNN technique for industrial control systems security. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 593–597 (2019). https://doi.org/10.1109/SMC.2019.8913895
11. Huang, G., Liu, Z., Weinberger, K.Q.: Densely connected convolutional networks. CoRR abs/1608.06993 http://arxiv.org/abs/1608.06993 (2016)
12. Jiang, W., Bruton, L.: Lossless color image compression using chromatic correlation. In: Proceedings DCC 1999 Data Compression Conference (Cat. No. PR00096), pp. 533 (1999). https://doi.org/10.1109/DCC.1999.785690
13. Kimmel, J.C., Mcdole, A.D., Abdelsalam, M., Gupta, M., Sandhu, R.: Recurrent neural networks based online behavioural malware detection techniques for cloud infrastructure. IEEE Access **9**, 68066–68080 (2021). https://doi.org/10.1109/ACCESS.2021.3077498

14. Lee, J.Y., Dernoncourt, F.: Sequential short-text classification with recurrent and convolutional neural networks. CoRR abs/1603.03827 http://arxiv.org/abs/1603.03827 (2016)
15. Lihao, W., Yanni, D.: 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), pp. 987–990, November 2018. https://doi.org/10.1109/ICSESS.2018.8663824
16. Liu, C., Dai, L., Cui, W., Lin, T.: A byte-level CNN method to detect DNS tunnels. In: 2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC), pp. 1–8 (2019). https://doi.org/10.1109/IPCCC47392.2019.8958714
17. Liu, G., Zhao, F.: An efficient compression algorithm for hyperspectral images based on correlation coefficients adaptive three dimensional wavelet zerotree coding. In: 2007 IEEE International Conference on Image Processing, vol. 2, pp. II - 341-II - 344 (2007). https://doi.org/10.1109/ICIP.2007.4379162
18. McDole, A., Abdelsalam, M., Gupta, M., Mittal, S.: Analyzing CNN based behavioural malware detection techniques on cloud IaaS. In: Zhang, Q., Wang, Y., Zhang, L.-J. (eds.) CLOUD 2020. LNCS, vol. 12403, pp. 64–79. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59635-4_5
19. Milton-Barker, A.: Inception v3 deep convolutional architecture for classifying acute (2019). https://software.intel.com/content/www/us/en/develop/articles/inception-v3-deep-convolutional-architecture-for-classifying-acute-myeloidlymphoblastic.html
20. Mobadersany, P., et al.: Predicting cancer outcomes from histology and genomics using convolutional networks. Proc. Nat. Acad. Sci. **115**(13), E2970–E2979 (2018). https://doi.org/10.1073/pnas.1717139115, https://www.pnas.org/content/115/13/E2970
21. Wang, Q., Shen, Y.: A jpeg2000 and nonlinear correlation measurement based method to enhance hyperspectral image compression. In: 2005 IEEE Instrumentationand Measurement Technology Conference Proceedings, vol. 3, pp. 2009–2011 (2005). https://doi.org/10.1109/IMTC.2005.1604524
22. van Wyk, F., Wang, Y., Khojandi, A., Masoud, N.: Real-time sensor anomaly detection and identification in automated vehicles. IEEE Trans. Intell. Transp. Syst. **21**(3), 1264–1276 (2020). https://doi.org/10.1109/TITS.2019.2906038
23. Zhang, Y., Chen, X., Jin, L., Wang, X., Guo, D.: Network intrusion detection: based on deep hierarchical network and original flow data. IEEE Access **7**, 37004–37016 (2019). https://doi.org/10.1109/ACCESS.2019.2905041
24. Zhao, X., Gao, L., Chen, Z., Zhang, B., Liao, W., Yang, X.: An entropy and MRF model-based CNN for large-scale landsat image classification. IEEE Geosci. Remote Sens. Lett. **16**(7), 1145–1149 (2019). https://doi.org/10.1109/LGRS.2019.2890996