Evaluation of Windows OS Security Using a Convolutional Neural Network: Part 3

Sujit Khanna
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14586
sk2913@rit.edu

I. Executive Summary

In this paper we present a novel way of evaluating the windows operating system security by using convolutional neural networks, instead of a standard multi-layered perceptron. A lot of challenges were involved in dealing with the input data generated from the expert system, as the data generated was highly imbalanced, which required us to develop our own custom built upsampling module. On top of that the biggest issue in using a convolutional neural network for this classification task was to represent the input data as an image. This was done by reshaping the row vector into a matrix that represents a monochromatic image. A majority of work in this paper deals with the model development and hyper-parameter optimization. A host of different filter numbers as well as different types of non-linear activation functions were tested in our grid-search hyper-parameter optimization technique. The paper also describes a few plots and classification reports to represent the strength of this technique, which achieves a state of the art classification performance on this classification task. To our knowledge such a technique has not yet been implemented in this domain.

II. Requirements

The previous part of the project dealt with creating a rule based expert system, that can evaluate the strength of windows operating system security. However the primary issue with a rule based expert system is that they require high computational cost, especially if there are multiple rules, that follow a hierarchy. Other than computational costs, expert systems do not have the ability to learn, as they follow a given set of *IF THEN* rules. This makes it very difficult to represent a highly non-linear input-output relation, with the help of such Boolean operators. For such an application, we need models that can effectively approximate the surface of the input-output function.

To tackle the issue of computational cost and poor (non-existent) learning ability of the rule based expert system, we approach this problem by implementing a machine learning algorithm for evaluating the windows operating system security score. Since machine learning is a broad term, that covers a host of sub-learning types (supervised, unsupervised, reinforcement learning etc.), as well a multiple algorithms falling into these categories, selecting an appropriate supervised learning algorithm becomes crucial. Though there are supervised machine learning models like kernel-SVM, Random Forest, and XGboost that can model non-linearity they often fail to perform up to the standards for highly non-linear problems. However a neural network with multiple hidden layers like Multi-Layer Perceptron, Convolution Neural Network or Recurrent Neural Networks are very apt at approximating any form of a function. Such neural networks are also called *Universal Approximators* for the same reason. In this paper we generate the input-output data-set required for supervised learning by running through all possible combination of inputs into the expert system and collecting the respective outputs. The choice of classification algorithm used in this paper is Convolutional neural network or CNN[1].

III. Implementation

To implement the our security evaluation framework as a CNN, we first had to generate the dataset from the expert system developed in the part 2 of the project. Our original expert system had 15 metrics/features, that were assigned a Boolean value based on predetermined thresholds, which were passed on to the expert system. To create this dataset we considered all possible combinations using the python's internal itertools library [2]. Iterating all 15 Boolean metrics ensure we had 2^{15} events or rows with input-output relation, as our expert system output a score between 0-10, we get a total of 11 classes/labels in the output. Since our choice of model is CNN, that works best with input data in a form of a square matrix, just like images we added a dummy variable/metric whose value is always 1, this is analogous to padding used in image classification.

Though the input metrics were discussed in great detail in the previous project, for the purpose of completeness we briefly describe these metrics again as follows,

- ***Ratio of allowed to dropped packets (ADR):*** This metrics calculates the ratio of allowed to dropped packets from the firewall logs, a very high value of this ratio indicates higher security risk. A high value of this ratio suggests an inadequate policy implementation to stop the operating system from external attacks.

- ***ICMP packets/seconds (icmp_rate):*** This metric indicates total number of ICMP packets allowed per second, a higher value indicates higher risk of a attack, for our expert system, we set this threshold to 2000 packets/seconds.
- ***TCP packets/seconds (tcp_rate):*** This metric indicates total number of TCP packets allowed per second, a higher value indicates higher risk of a attack, for our expert system, we set this threshold to 8000 packets/seconds.
- ***Enabled All Firewall Profiles (EAFP):***This metric checks if all firewall profiles such as domain profile, private profile, and public profile are enabled. If all profiles are enabled the security risk is low, if they are not the security risk is high.
- ***Advanced IPSec Configurations (AIpC):*** This metric checks if advanced IPsec settings are enabled. If Key exchanges and Data Protection services are set to advanced the security risk is low, and if they are not the security risk is high.
- ***Blocking Outbound Connections (BOC):*** This metric checks if there are any specific rules set to block services from communicating on untrusted profiles and to non-private resources. Specifically this rule checks if following executable files are blocked from communicating with any network resources. *Mshta.exe* which is used by attackers to proxy execution of malicious files, *Cscript.exe* which is used by attackers to execute malicious scripts, and *Wscript.exe* which is also used by attackers to execute malicious script [3]. If all three *.exe* files are blocked from communicating with any network resource the system security risk is low.
- ***Event IDs Set 1 (EventSet1):***According [4] to Event IDs 5156,5158, and 5154 indicate network communication of the operating system and running applications and can greatly improve the detection of security threats. Hence higher number of these event IDs in the events logs would indicate a high system security vulnerability.
- ***Event IDs Set 2 (EventSet2):*** Event ID 4625 indicates a failed logon attempt a large number of these events may be predictive of a password hacking or password spraying attacks.
- ***Event IDs Set 3 (EventSet3):*** Event ID 6006 logs the events when the system was shutdown or restart, a large occurrence of these events maybe indicative of malicious attempts to avoid logging of the activity.
- ***Event IDs Set 4 (EventSet4):*** Event ID 7034 indicates when a service is terminated unexpectedly, again a large number of these events in the event log may be indicative of an attack on the security system
- ***Browser Auto-update (BAP):*** This metric checks if the browser's auto-update feature is enabled. A disabled auto-update feature in the web-browser may cause system vulnerabilities as it may be susceptible to new malware or virus attacks.
- ***Unwanted Plugins Disabled (UPD):*** This is a boolean user input that indicates if it has disabled the unwanted plugins. As unwanted plugins can cause virus attacks, the system is more secured if all unwanted plugins are disabled
- ***Pop-ups blocked (PUB):*** In some cases adware programs can generate malware pop ups that can infect the browser and attack the system security, hence the security of the system is more robust if the pop-ups in the web browser are blocked.
- ***phishing and malware protection enabled (PMPE):*** This metrics checks if phishing and malware protection is enabled in the privacy setting of google chrome browser. This metric warns if the web-page the user is visiting contains malware or is a phishing attack.
- ***Password_Strength:*** This metric checks weather the password strength of the system is strong enough to based on predefined functions

A. Data Preprocessing

One of the most common problems when dealing with multi-label classification is the issue of class imbalance. The class imbalance problem was stark in the data generated by the initial expert system. The image below shows the count of each label in the dataset, where labels 0, 1, 2, 3 constituted majority of classes, and the rest of the labels were highly misrepresented in the dataset. Having such a highly imbalanced class data, makes the classifier overfit on a few classes, hence these under-sampled classes must be upsampled. Our upsampling procedure, checks if the total count of any

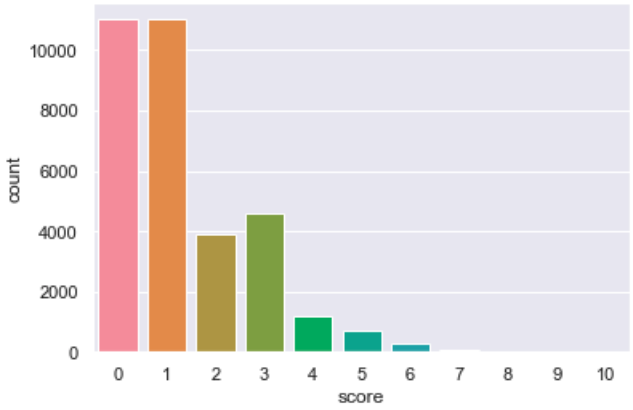

Fig. 1. Count of Classes: Imbalanced Case

label/class is less than 10%, and copies the existing batch of labels till that particular class label's count is $\geq 10\%$. The results after applying this upsampling procedure can be seen below.

The next step of preprocessing the dataset involves converting the conventional numeric dataset into an matrix form that follows a representation of an image. In this case we convert the row vector into a matrix. For example an array

$$[1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0]$$

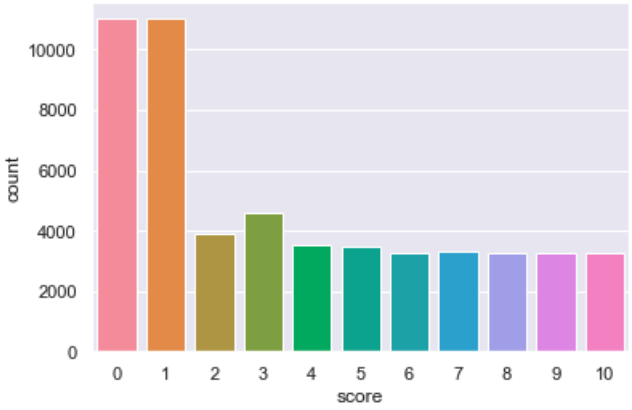

Fig. 2. Count of Classes: Balanced Case

is converted into a matrix

$$[[1,1,1,1],[0,0,0,0],[1,1,1,1],[0,0,0,0]]$$

. The sample figures below shows how this relation is presented to the convolutional neural network as an image to be classified, here the black pixel assumes a value of 1, and white pixel assumes a value of 0. In this case the output (security score) is 6 and input matrix is

$$[[1,1,1,1],[1,1,0,],[1,0,1,0],[1,0,1,1]]$$

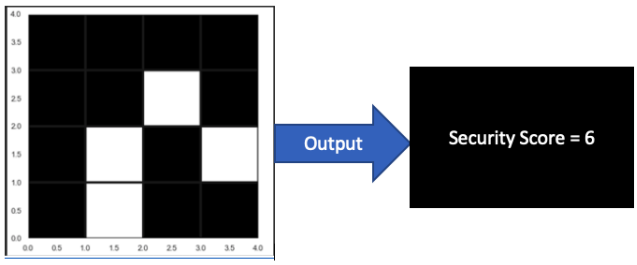

Fig. 3. Conversion of inputdata for CNN

B. Model Implementation

This sub-section details the aspects of the model implementation, which includes the choice of libraries for model implementation and data preprocessing as well as other software and hardware requirements.

1) Machine Learning Specifications: Our machine learning pipeline has two components, the data processing component and the model building component.

- *Data Preprocessing:* The data preprocessing is primarily carried out by the famous pandas library [5], as well as scikit-learn library that offers a host of standard machine learning APIs, and functionalities [6]
- *Model Building:* For the model building component we used keras [7], which is high level library built on tensorflow. The reason we chose keras was its high adaptability, ease of use and range of functionalities.

2) Non Machine Learning Software Requirements:: The primary Non Machine Learning software requirement is to have a windows 10 operating system installed on a laptop on which the application will run. Since the entire application is built on python, the application needs installation of python3.7. This is a strict requirement as no other python version will be compatible with this application. The python environment also requires a host of libraries to be installed for this application. These libraries are included in the *requirements.txt* file in the project, which are *experta* [8], *pandas* [5], *WMI* [9], and *password_strength* [10]. Another important requirement for ensuring the application runs smoothly is to enable firewall logging[11], as in the default windows setting this logging is disabled. In case the logging is disabled the default metric values are all assigned the *false* label as a strict assumption in our expert system.

3) Hardware Requirements: The only major hardware requirement is a windows laptop, with windows 10 installed on it. Decent RAM size and processing power is desirable as the application is a little data intensive, since it needs to parse through a host of logs to generate the raw baseline metrics.

IV. Experiments

This section provieds details about the experimental analysis done in the project, which will primarily focus on the hyper-parameter optimizations, and other model specification used in analysing the machine learning model. In our experiments we kept a few configurations fixed, as they're necessary for training our model, and other configurations vary, that are a part of the hyper-parameter optimization module required to fine tune the model.

A. Fixed Parameters

The fixed model parameters in our analysis do not change with different hyper parameter configuration, and are detailed below

- *Loss Function:* Since our problem is that of a multi-label classification that involves accurately classifying 11 labels, the choice of our loss function is *categorical_crossentropy*. This is similar to binary entropy information but with more variables, the lower the entropy the higher the information, hence as a loss function this must be minimized
- *Optimizer:* The choice of optimizer in our case is *ADAM* which is a form of adaptive stochastic gradient optimizer whose optimization path depends on the first order and second order moments. This optimizer is much more sophisticated than other optimizers such as *Stochastic Gradient Descent*, and *RMS_Prop*, and has been shown to consistently perform well for classification problems
- *Kernel and Maxpool Size:* Since our input data matric is a 4x4 matrix, we have to limit the size of the filter and maxpool layers, therefor the kernel size is (2 x 2), and maxpool size is (1 x 1).

- *Train, Validation and Test Split:* For our experiments we performed a train-validation-test split in the ratio of 60-20-20. We had a larger size of the validation set to ensure there's symmetry in the sizes of the validation and test datasets. This is helpful in avoiding overfitting on the training set.
- *Dropout:* The dropout layer has been added just before the the fully connected layer, to avoid overfitting. Since our data is fairly standard, we added a low dropout probability of 0.5.
- *Number of Layers:* In our model we went ahead with two modules of conv2D-maxpool layers, followed by a fully connected layer with softmax as the activation function for classifying the network output. The image below provides a sample network description with 64 filters in the first layer, and 128 filters in the second layer. Hence we can consider the Convolutional Neural Network as having 2 hidden layers.

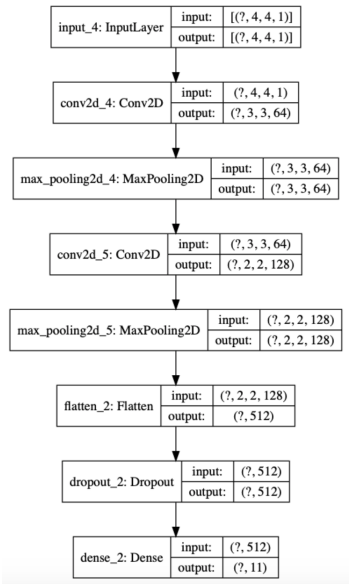

Fig. 4. Sample CNN Model Representation

B. Hyper Parameters Optimization:

The two major components on which we could use hyper parameters were the number of filters in each layer, as well as the type of activation functions used in the analysis.

- *Activation Functions:* To test the effectiveness of a decent size of activation functions to our problem, we choose a list of activation functions which were permuted for all the configurations of the filter sizes, these activation functions were
  - *RELU: Rectified Linear Unit Activation*
  - *ELU: Exponential Linear Unit Activation*
  - *Sigmoid: Standard Sigmoid Activation*
  - *Tanh: Hyperbolic Tangent Activation*
  - *SELU: Scaled Exponential Linear Unit Activation*
- *Number of Filter:* The number of filters determine is analogous to number of neurons as they add to the depth of the network, a larger number of filters, will stack together larger processed outputs for the next layer. Usually the state of the art performance in CNNs is obtained by using a high number of filters. In our model, the number of filters in the second layer will always be twice of the first layer, hence we only need to grid search one set of filter number configuration. The list below shows the number of filters tested in our paper. $[8, 16, 32, 64, 128]$

V. Results

In this section we provide details about the results of different hyper parameter optimization configuration, as well as the training details and classification report on the best available configurations. Since we used 5 different configurations of activation function and 5 different nnumber of filters, we had about 25 different results to analyze.

A. Hyper-Parameter Optimization

We analyzed the 25 different models on a host of metrics, these metrics are briefly described below.

- *param_config:* This field indicates the configuration of activation function and filter number, i.e. a run identifier
- *train_loss:* This field indicates the training set categorical crossentropy loss
- *validation_loss:* This field indicates the validation set categorical crossentropy loss
- *test_loss:* This field indicates the test set categorical crossentropy loss
- *train_accuracy:* This field indicates the training set multi-label classification accuracy
- *validation_loss:* This field indicates the validation set multi-label classification accuracy
- *test_loss:* This field indicates the test set multi-label classification accuracy
- *run_time(sec):* This field indicates the training time in seconds
- *memory_consumption(MegaBytes):* This field indicates the memory resource consumed during the training process

The figure below shows the above metrics for different hyper-parameter configuration.

Based on the above table we can see the four best configurations are config1: *(num_filters = 64, activcation_func = elu)*, config2: *(num_filters = 64, activcation_func = relu)*, config3: *(num_filters = 128, activcation_func = relu)*, and config4: *(num_filters = 16, activcation_func = relu)*. Based on these configurations we see that even for different configurations of filter numbers *relu* performs better than other activation functions.

B. Best Configurations

Apart from generic analysis done on the hyper-parameter runs, which looked at a set of metrics, in this subsection,

param_config	train_loss	validation_loss	test_loss	train_accuracy	validation_accuracy	test_accuracy	run_time (sec)	memory_consumption(MB)
num_filters = 64, activcation_func = elu	0.04	0.03	0.03	0.98	0.99	0.99	38.25	537.85
num_filters = 64, activcation_func = relu	0.04	0.03	0.03	0.99	0.99	0.99	33.69	526.10
num_filters = 128, activcation_func = relu	0.03	0.03	0.04	0.99	0.99	0.99	63.33	596.58
num_filters = 16, activcation_func = relu	0.07	0.05	0.05	0.97	0.98	0.99	19.78	412.69
num_filters = 32, activcation_func = relu	0.04	0.04	0.04	0.99	0.99	0.99	23.61	466.69
num_filters = 32, activcation_func = tanh	0.05	0.05	0.04	0.98	0.99	0.99	24.59	492.22
num_filters = 128, activcation_func = tanh	0.03	0.04	0.04	0.99	0.99	0.99	68.93	625.71
num_filters = 32, activcation_func = elu	0.06	0.04	0.04	0.98	0.98	0.98	25.71	478.65
num_filters = 32, activcation_func = selu	0.06	0.04	0.04	0.98	0.98	0.98	24.71	516.69
num_filters = 64, activcation_func = selu	0.05	0.04	0.04	0.98	0.98	0.98	35.27	577.92
num_filters = 128, activcation_func = elu	0.04	0.04	0.03	0.99	0.98	0.98	70.07	610.43
num_filters = 128, activcation_func = sigmoid	0.04	0.05	0.04	0.99	0.98	0.98	65.48	629.70
num_filters = 16, activcation_func = elu	0.08	0.05	0.05	0.97	0.98	0.98	20.57	424.89
num_filters = 64, activcation_func = sigmoid	0.06	0.05	0.05	0.98	0.99	0.98	35.09	564.16
num_filters = 64, activcation_func = tanh	0.04	0.04	0.04	0.99	0.99	0.98	34.62	551.15
num_filters = 128, activcation_func = selu	0.05	0.05	0.05	0.98	0.98	0.98	65.28	643.09
num_filters = 16, activcation_func = tanh	0.10	0.07	0.07	0.97	0.98	0.98	19.25	438.42
num_filters = 16, activcation_func = selu	0.11	0.06	0.06	0.96	0.98	0.98	19.86	451.83
num_filters = 8, activcation_func = relu	0.13	0.08	0.08	0.95	0.98	0.98	24.55	357.22
num_filters = 32, activcation_func = sigmoid	0.11	0.09	0.10	0.97	0.97	0.98	24.71	504.93
num_filters = 8, activcation_func = elu	0.15	0.10	0.10	0.95	0.97	0.97	25.80	362.06
num_filters = 8, activcation_func = selu	0.18	0.12	0.12	0.93	0.96	0.96	17.26	400.90
num_filters = 8, activcation_func = tanh	0.20	0.15	0.15	0.93	0.96	0.96	19.35	374.28
num_filters = 16, activcation_func = sigmoid	0.21	0.17	0.18	0.93	0.96	0.96	19.81	442.19
num_filters = 8, activcation_func = sigmoid	0.38	0.33	0.33	0.87	0.90	0.90	17.58	387.56

Fig. 5. Results of Hyper-Parameter Optimization

we take a deeper look at the top 4 configurations, as well as analyze the classification report of the best configuration.

1) Training/Validation Accuracy and Loss: We will analyze plots that describe how the training/validation accuracy/loss evolves with the number of epochs (iterations). These details pertaining to the 4 parameter configurations are shown below.

- *config1: (num_filters = 64, activcation_func = elu):* Based on the figure below we see that training and validation loss is minimized fairly quickly and then proceeds to decrease in smaller steps after 20 epochs. This shows that this particular configuration achieves local minima fairly quickly. The figure 7 shows the evolution of training

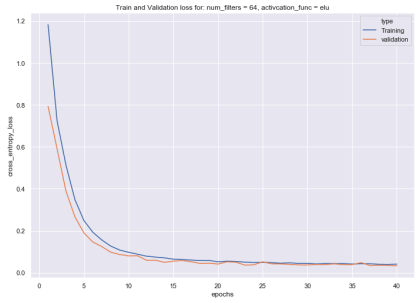

Fig. 6. Parameter Config 1: Training and Validation Loss

and validation accuracy with epochs, the accuracy also follows a similar pattern where the highest improvement in the accuracy is achieved before 20 epochs, and minor improvements are observed in succeeding epochs. An important consideration over here is that there is no deviation between training and validation accuracy/loss metrics. This indicates that there is very less chance of overfitting on the input data.

- *config2: (num_filters = 64, activcation_func = relu)* The training and validation loss with respect to epochs for this configuration is very similar to the previous configuration. This should not be surprising as they only differ in the choice of activation function.
  Similar to the previous configuration, even the training and validation accuracy seems to evolve in a similar fashion with respect to number of epochs.
- *config3: (num_filters = 128, activcation_func = relu):* This specific configuration, has the highest number of

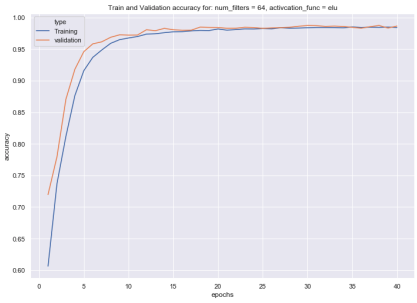

Fig. 7. Parameter Config 1: Training and Validation Accuracy

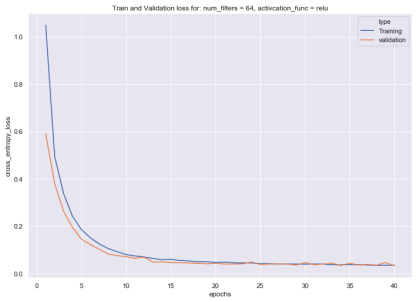

Fig. 8. Parameter Config 2: Training and Validation Loss

filters at 128, hence we observe the local minima at about 15 epochs, this is a large improvement from the previous configurations, however the difference in the final loss metrics after 40 epochs is negligible.
The training and validation accuracy follows a similar pattern to the loss configuration, where the maximum accuracy is achieved in around 15 epochs, the training and validation curves in this configurations do not deviate as well, indicating a lower probability of overfitting

- *config4: (num_filters = 16, activcation_func = relu):* This particular configuration has the fewest number of filters at just 16, and unsurprisingly we see that the categorical cross entropy loss function follows a gradual minimization path unlike the previous configurations. Even after 40 epochs the loss function is not as low as configurations of the previous models. Even in case of the accuracy plot, the highest accuracy is not achieved around 20 epochs as seen in previous epochs, the lower number of filters seem to adversely affect the accuracy of the convolutional neural network.

2) Classification Report: Config 1: This subsection details the classification report generated by the *config1: (num_filters = 64, activcation_func = elu)* using sklearns *classification_report* function [12]. This function generates two tables, one containing label-wise precision, recall, f1-Score and support, as well as different averaged metrics. The table below shows the label wise precision, recall, f1-score and Support (which indicates the number of occurrences of each label) for the test set predictions. Looking at the table it is evident that the model performs really well on this balanced dataset, as

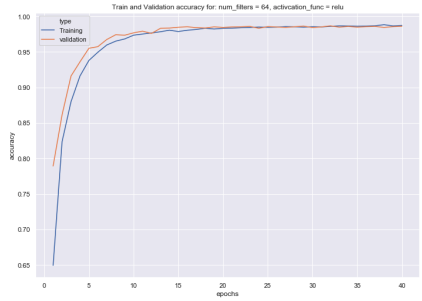

Fig. 9. Parameter Config 2: Training and Validation Accuracy

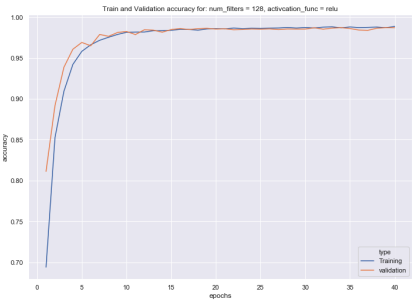

Fig. 11. Parameter Config 3: Training and Validation Accuracy

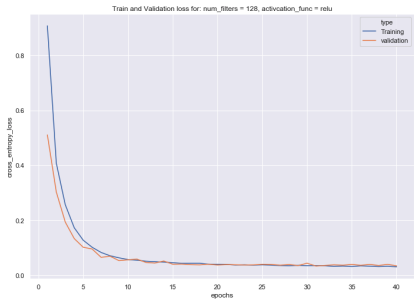

Fig. 10. Parameter Config 3: Training and Validation Loss

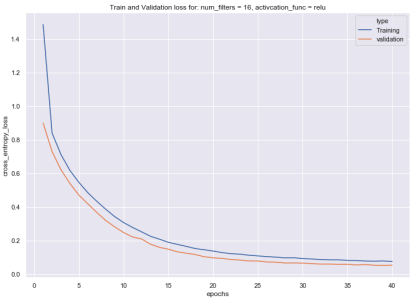

Fig. 12. Parameter Config 4: Training and Validation Loss

the precision, recall and f1-score are all above 0.96 for all the class labels, such a high number indicates the strength and stability of the classifier. The most impressive feature of the model is that it has the best possible scores for the extreme classes, which is critical, as these can be considered as low frequency and high severity events, and can cause a significant impact if it gets these predictions wrong.

TABLE I
CLASSWISE PRECISION, RECALL, F1-SCORE AND SUPPORT

Class Label	Precision	Recall	F1-Score	Support
0	0.99	0.99	0.99	2218
1	0.99	0.99	0.99	2298
2	0.96	0.98	0.97	743
3	0.97	0.97	0.97	920
4	0.99	0.97	0.98	680
5	0.99	1.0	0.99	705
6	0.98	0.99	0.98	656
7	1.0	1.0	1.0	654
8	0.99	1.0	0.99	690
9	1.0	1.0	1.0	658
10	1.0	1.0	1.0	669

The previous table detailed class wise classification metrics, but analyzing such high dimensional table can make it difficult to assess the overall performance of the classifier. Therefore we have three averaging schemes that can compress the overall results, the three averaging schemes are

- *micro average:* It computes the average of false-positives, false-negatives and true-positives for all classes.
- *macro average:* It computes the unweighted means for each label, i.e. it takes the mean of the outputs for each label, and takes the average of these means
- *micro average:* It is similar to *macro mean* however it computes the average support weighted mean per label.

Based on the table below, we see that all averaging schemes indicate a really impressive classifier performance, which is in line with what's observed in the previous table.

TABLE II
AVERAGED PRECISION, RECALL, F1-SCORE AND SUPPORT

Average Metric	Precision	Recall	F1-Score	Support
micro average	0.99	0.99	0.99	2218
macro average	0.99	0.99	0.99	2298
weighted average	0.96	0.98	0.97	743

VI. USER GUIDE

This section details how one should install and use this application. Some necessary prerequisites for running the application are show below.

- *Installation of Python 3.7:* Our project is built on python3.7[13], hence it's is paramount to the execution of the application that this version of python is installed in the user's Microsoft Windows 10 operating system (Also make sure that python3.7 is added the the PATH)
- *Ensure Firewall logging is Enabled:* By default in most windows operating system, windows firewall logging is disabled. To ensure the application considers the firewall logs for the metrics enable the firewall logging using the following link [11].

A. Run the System application

In this subsection we depict how one can run the trained CNN model on the windows operating system, without having to go through the training process.

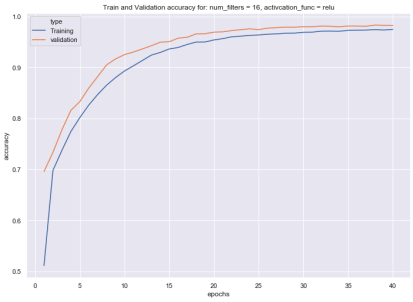

Fig. 13. Parameter Config 4: Training and Validation Accuracy

- *Step 1:* After unzipping the zip folder, please navigate to the folder named *pr3_sujit_khanna_codebase* inside the folder *pr3_sujit_khanna*
- *Step 2:* Open windows command prompt as admin. Note it's important to open the command prompt as admin to ensure that application runs. Since the scripts need special permission, without the admission privileges, the script fails. After opening the command prompt *CD* into this folder location, for example
  cd C:/Users/username/pr3_sujit_khanna

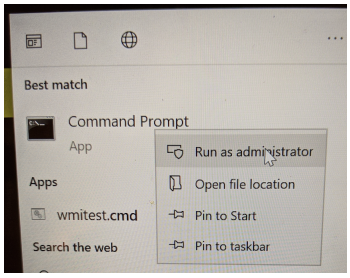

Fig. 14. Run Windows Command Prompt as Admin

- *Step 3 (Install Requirements):* The next step involves installing all the prerequisite libraries required to execute the application. This can be done by executing the command below in the command prompt
  pip3.7 install -r requirements.txt
- *Step 4 (Run the Application):* This step involves running the main python file containing our trained CN model. The command to run the application on the command prompt with *admin* privileges is
  py -3.7 pr3_sujit_khanna.py
- *Step 5 (User's Input):* Similar to step 4 in the previous subsection, the user's input are case sensitive, and the inputs must either be *y* or *n*, except in case of the users password input.

B. Run Analysis Jupyter Notebook

In this section we describe the basic steps in starting the jupyter notebook.
One should navigate to the main project folder mentioned before i.e. *pr3_sujit_khanna_codebase* and execute the following command in the windows command prompt running as administrator.
jupyter notebook CNN_Multiclass_Classification_.ipynb
This commnad will open up a jupyter notebook on the default web browser, and one can execute all the cells, for all the analysis by pressing the restart and run all butto in the kernal tab, as shown in the image below.

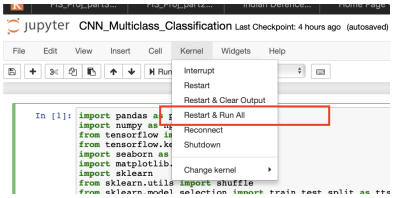

Fig. 15. Running all cells of the jupyter notebook

For the purpose of simplicity we've provided a text file called *instructions.txt* with commands to execute all the scripts in the *pr3_sujit_khanna_codebase* folder.

REFERENCES

[1] "Cnn," https://en.wikipedia.org/wiki/Convolutional_neural_network.
[2] "itertools," https://docs.python.org/3/library/itertools.html.
[3] D. Stuckey, "Endpoint isolation with the windows firewall," September 2015, https://medium.com/@cryps1s/endpoint-isolation-with-the-windows-firewall-462a795f4cfbhttps://medium.com/.
[4] J. Baráth, "Optimizing windows 10 logging to detect network security threats," in *2017 Communication and Information Technologies (KIT)*, 2017, pp. 1–4.
[5] "Pandas library," May 2020, https://pandas.pydata.org.
[6] "sklearn," https://scikit-learn.org/stable/index.html.
[7] "keras," June 2020, https://keras.io.
[8] "Experta library," November 2019, https://github.com/nilp0inter/experta.
[9] "Wmi library," April 2020, https://github.com/tjguk/wmi.
[10] "password strength library," Jan 2019, https://pypi.org/project/password-strength/.
[11] "Firewall logging," August 2017, https://docs.microsoft.com/en-us/windows/security/threat-protection/windows-firewall/configure-the-windows-firewall-log.
[12] "sklearn classification report," https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html.
[13] "python3.7.7," March 2020, https://www.python.org/ftp/python/3.7.7/python-3.7.7-amd64.exe.
[14] N. Naik, P. Jenkins, N. Savage, and V. Katos, "Big data security analysis approach using computational intelligence techniques in r for desktop users," in *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2016, pp. 1–8.
[15] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Comput. Surv.*, vol. 49, no. 4, Dec. 2016. [Online]. Available: https://doi.org/10.1145/3005714
[16] "Securing your web browser," September 2015, https://www.us-cert.gov/publications/securing-your-web-browserGoogle.
[17] "Windows event log analysis," September 2018, https://www.forwarddefense.com/pdfs/Event_Log_Analyst_Reference.pdf.