Intelligent and Converged Networks **ISSN 2708-6240**
2021, 2(1): 66–82 **DOI: 10.23919/ICN.2020.0023**

CNN and MLP neural network ensembles for packet classification and adversary defense

Bruce Hartpence* **and Andres Kwasinski**

Abstract: Machine learning techniques such as artificial neural networks are seeing increased use in the examination of communication network research questions. Central to many of these research questions is the need to classify packets and improve visibility. Multi-Layer Perceptron (MLP) neural networks and Convolutional Neural Networks (CNNs) have been used to successfully identify individual packets. However, some datasets create instability in neural network models. Machine learning can also be subject to data injection and misclassification problems. In addition, when attempting to address complex communication network challenges, extremely high classification accuracy is required. Neural network ensembles can work towards minimizing or even eliminating some of these problems by comparing results from multiple models. After ensembles tuning, training time can be reduced, and a viable and effective architecture can be obtained. Because of their effectiveness, ensembles can be utilized to defend against data poisoning attacks attempting to create classification errors. In this work, ensemble tuning and several voting strategies are explored that consistently result in classification accuracy above 99%. In addition, ensembles are shown to be effective against these types of attack by maintaining accuracy above 98%.

Key words: Convolutional Neural Network (CNN); Multi-Layer Perception (MLP); ensemble; classification; adversary

1 Introduction

Accurately identifying packets as they enter or pass a particular point in a network is critical to most operations, including security applications, quality of service, and management. Traditionally this task has fallen to parsers examining header data or rule based systems checking traffic against lists that permit or deny traffic. With the advent of greater processing power and increased memory, machine learning techniques are receiving much more attention by researchers and industry alike. Among them, neural networks show great promise, and several types have been investigated for these applications. In preludes to this work, both Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN) models were found to be extremely effective for the base task of packet classification[1, 2].

However, a single neural network (or any machine learning based classifier) may fall short of accuracy targets and thus may be of limited use for more advanced tasks such as attack recognition or quality of service problem identification. In these cases, it is beneficial to deploy a collection of machine learning classifiers in the form of an ensemble so that their results might be compared and a more accurate classification can be made. An ensemble or committee is a collection of redundant machine learning models of the same or different types. They are considered redundant because each of them present a solution to the same task[3]. One goal in creating ensembles is to ensure some level of diversity such that the models do not arrive at the same errors or misclassification problems. In this way, voting

• Bruce Hartpence is with the GCCIS i-School at the Rochester Institute of Technology, Rochester, New York, NY 14623, USA. E-mail: bhhics@rit.edu.
• Andres Kwasinski is with the Department of Computer Engineering, KGCOE at the Rochester Institute of Technology, Rochester, New York, NY 14623, USA. E-mail: axkeec@rit.edu.
∗ To whom correspondence should be addressed.
Manuscript received: 2020-09-03; revised: 2020-10-12; accepted: 2020-12-08

on independent model results can arrive at the correct label for the sample[4].

Neural network ensembles have been utilized in many fields, including image recognition and health sciences[5]. They have also been used for food recognition and tree classification[6, 7]. In these cases, ensembles have been shown to improve results, but their efficacy for problems in communication networks is poorly understood. The ensemble investigated here utilizes several stages to classify packets in general categories and then moves on to User Datagram Protocol (UDP) and Transmission Control Protocol (TCP) specific classes. When leveraging ensembles, researchers may choose to deploy a variety of techniques. For example, the ensemble might be made up of decision trees, neural networks, and support vector machines. Another approach is to use similar classifiers but modify their operation and behavior in order to examine the features from slightly different perspectives. It is the latter methodology that was adopted for the results described here.

The MLP and CNN models that form this ensemble redundant yet distinct so that they can successfully vote on the resultant high probability class labels. The models are differentiated by the number of input features, optimizers, and the internal structure of the various neural networks.

The output of an ensemble is a collection of labels of a particular sample. In this case, each member of the ensemble attempts to classify each packet in the dataset. Ensemble members rarely agree on the correct label. As such, a technique for combining these answers must be used. This voting can be accomplished in a variety of ways and several are explored in this article. What is important to note is that the ensemble and the voting methodology can be application specific. Simple majority voting can be successful, but a weight vote may be more effective. Majority voting is often considered a special case of weighted voting where the weight of a vote from each model is 1[5, 8]. In this work, model dropout, majority voting, validation set weighting, and overall performance based weighting are investigated.

Neural network ensembles can also be used to defend machine learning systems against adversarial attacks or perturbations. The adversary injects a type of noise during system input which causes misclassification errors[9]. These errors can have serious ramifications in systems such as self-driving cars or network security systems. Ensembles have been shown to be effective against small perturbations. Larger changes to input data can require a modification to the training process which must now include perturbations similar to the attach profile[10].

The following contributions are provided in this study:

• An effective structure and tuning methodology for packet classification ensembles;

• An investigation on effective voting strategies; and

• Exploration on ensemble effectiveness against several adversarial attacks.

In the following sections, the architecture of the system is described along with the datasets and the processing necessary before datasets as input into the system described. The tuning process and ensemble voting are also covered in detail. Lastly the performance data for the adversarial attacks are discussed.

2 Related work

Machine learning models and the ensembles built from their combination have been effective for many applications. In the tree classification problem[6], aerial images of trees are input into an ensemble of CNNs and random forests, as a results, accuracy significant increases in through a weighted voting strategy. For food recognition[7], a collection of established models, including ResNet[11] and VGGNet, are used[12]. In this case, a variety of voting strategies compared with promising results.

The use of weighted voting can also be found in Ref. [13] in which the ensemble examined hyper-spectral images from a variety of feature sets. This approach is shown to be effective on available datasets. The effect of protein mutations is studied in Ref. [14]. The results show that the ensemble approach and voting yield a 4% increase in accuracy for these challenging datasets. In a very timely example, Bitcoin trading was predicted[15] using single model, trend following, and ensemble approaches. The ensemble outperformed the others with prediction accuracy of 58%–63% and a profit of 85%.

Many of these works do not provide performance improvement values owing to the works being in progress or because measurements used are in terms of loss or error. However, all of these researcher agree that the ensemble approach improves performance and is an important part of the strategy going forward. The best voting methodology is not always clear either. Majority voting certainly features prominently, but the weighting strategy efficacy is more difficult to evaluate because there are a variety of choices upon which to build the weighting strategy. For example, Ref. [7] includes six variations: minimum probability, average of probabilities, median, maximum probability, product of probabilities, and weighted probabilities. In Ref. [8] voting uses a weight vector that is created from the accuracies on each class. Their approach ranks highest in a majority of tests and reaches 88% accuracy.

While CNNs and MLPs have been applied to communication network challenges, there are fewer instances of ensembles created using neural networks. One example can be found in Ref. [16]. For example, neural networks are utilized to classify network traffic. They employ a weighted voting scheme and report an accuracy above 96%. Forthermore, an exploratory study[17] has attempted to address the class imbalance problem using neural network ensembles.

Other works utilize decision trees and more traditional algorithms in their ensembles. In Ref. [18], a collection of decision trees was evaluated. The authors noted that while the problem of increased performance time with multiple models may make ensembles impractical, there is little question that all of the ensembles tested out-perform single model systems. Reference [19] used an ensembles with separate classifiers to determine if the correct protocol is utilizing a particular port though it is not clear what the individual classifiers are. Markov chains are used in an weighted ensemble to classify encrypted web traffic. Many of these use available datasets such as CAIDA[20] for their research. These datasets can be somewhat dated and application specific. For these reasons, the work reflected in this article utilizes contemporary traffic captured on an operating testbed.

As noted earlier, attacks against machine learning architectures often use perturbations or modifications in dataset samples in order to create misclassification problems. It has been found that these perturbations extend across models[21]. As a result, defenses against these types of attack have been explored. In Ref. [22], a case is made for both adversarial samples being introduced into the training data and several models being trained independently. The authors[22] reported advancement on a variety of attacks including simple manipulation of the data and more complex attacks designed to disrupt the model loss function calculation. It should be noted that most of the adversarial studies have been done on image classification and specific attacks aim at modifying simple (such as might be seen with the human eye) and complex image properties.

In Ref. [10], we see a listing of the attacks and the argument for attack independent training as a system is only effective on known attacks. The authors[10] created several input transformations on ImageNet to create their ensemble. In the course of defending against several attacks, success rates are as high as 92%.

But what of attacks against communication traffic classifier? There are several traditional attacks against communication systems and many of them involve a modification of the data. Some attacks attempt to penetrate firewalls by passing themselves off as benign traffic types. Others attempt to hobble network defense by sending small, under-sized packets. Others also seek to pass traffic via some sort of covert channel or by pretending to have a different source or destination. In addition, encrypted data can create problems for legitimate systems and attackers. In Ref. [23], the authors showed that like models used for image processing, deep neural networks for communication systems are susceptible to adversarial samples. Most studies being done to combat this trend have only been published in the last one to two years. For these reasons, this work explores the idea of image modification and packet modification by making several changes to the packet headers, altering the protocol identification fields, and either zeroing or modifying other header data.

3 Architecture

The architecture was eventually comprised of six neural networks with differing configurations. In addition, there are several stages that focus on the differing categories of traffic. However, arriving at the final build and formulating the input data for use by the system represent the majority of the work.

Each stage has its own parser for the datasets to be used at that point. This includes data manipulation for all models and establishing the ground truth labels. This stage is followed by the models themselves, which have unique builds. This also means that the input data must be formulated based on a particular model's requirements. Each stage also has its own training data. Subsequently the models vote on the best class choice for each sample and make the predictions. An overview of the system is shown in Fig. 1.

Portions of each stage are now described in detail, beginning with the datasets.

3.1 Datasets

The datasets are constructed from network traffic seen in a local testbed as using external, non-contemporary traffic datasets is problematic. Laying the initial groundwork was 28 of the most common general protocols by percentage. These included categories such as spanning tree and many variations of IPv4-based traffic. While a high level of accuracy was achieved using MLP models for classification, subsequent work made it clear that a single model might have difficulty with the variations in traffic. It is also challenging to collect the necessary training samples for each possible class. For example, when addressing the problem of port scans, categorization of either TCP or UDP is not sufficient. Clearly there are the variations in target ports and IP addressing. In the case of TCP, there are the various flags and their combinations to be considered. This realization was the genesis of the sequential neural network models used here. The number of general categories is reduced and in the first stages, TCP and UDP are separated out so that dedicated models can be applied to them individually. Due to the significant difference in the headers, IPv6 was removed from the current testing. Models would be trained to recognize packets of a particular type only to be fooled into misclassification because the IPv6 data were encoded differently.

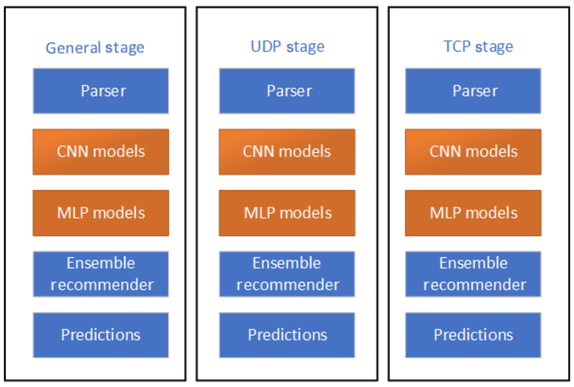


Fig. 1 Architecture overview.

There are seven datasets presented here: one training, one validation, and five test datasets. The training dataset is also used later in the process as the UDP and TCP trainers are derived from this population. Each class has 5000 samples for training. For practicality, this number is limited to eighteen classes. There are 10 general classes, two of which are TCP and UDP. Once the TCP and UDP packets are separated, they are further divided into the 3 TCP classes and 8 UDP classes. There is overlap in the totals because the general classes include TCP, UDP, and an empty class. Thus the general training datasets contains 90 000 samples. The validation set has 500 samples of each class pulled from the same population of packets used for the training sets. The test datasets are random collections of packets, though they have the same set of classes. These test datasets vary in size from $3.3{\times}10^3$ to 10^5 packets. Larger datasets of up to 2×10^5 packets were tested, however, their creation and testing are time consuming and provide similar results. Thus, these datasets were not included due to the sheer number of tests that would be run. Table 1 provides the breakdown of the class structure where the protocols included are Spanning Tree Protocol (STP), Cisco Discovery Protocol (CDP), Address Resolution Protocol (ARP), Internet Control Message Protocol (ICMP), Internet Group Management Protocol (IGMP), and Real Time Transport Protocol (RTP).

With a parser or analyzer such as Wireshark, classes are distinguished by fields values found at layer two,

Table 1 Breakdown of the class structure.

General	TCP	UDP
Empty	Port 443	DNS
STP	Port 80	DHCP
CDP	Port 8080	SSDP
Loopback	–	NBNS
ARP	–	RTP 1
ICMP Echo Req	–	RTP 2
ICMP Echo Reply	–	RTP 3
IGMP	–	RTP 4
UDP	–	–
TCP	–	–

three, and four. For example, ARP and IPv4 traffic vary in the Ethernet type code with ARP using 0x0806 and IPv4 using 0x0800. At layer 3, IPv4 traffic is further broken down by the protocol ID field value found in the IP header: ICMP is identified by a value of 1, IGMP is 2, TCP is 6, and UDP is 17. The custom parser built for this work utilizes the same technique to establish the ground truth. However, it is not applied to the neural network. Instead, the artificial neural network learns to identify these important features via the error calculation and weight update. In the case of the convolutional neural network, these features are re-imagined as part of an image and recognized as such. The UDP and training sets are actually part of the general trainer although the non-UDP and non-TCP packets are removed. They are structured in the same way with the UDP trainer having 40 000 samples (8 × 5000) and the TCP trainer having 15 000 samples. The UDP and TCP stages also have validation sets.

Previous work[1,2] has shown that both MLP and CNN models can be effectively used for classification efforts and serve as a basis for a variety of applications. What was discovered here is that for various stages of the architecture, models of a particular type perform better than others. For example, the MLP models work well in the general classification task; however, the CNN models tend to out-perform them for UDP classification.

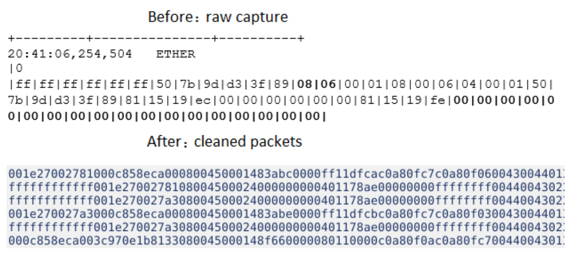
3.2 Preprocessing

Preprocessing is a part of most machine learning research. Incoming data is in a very raw form and often has artifacts that are of little or no use. In this case, Wireshark captures are obtained from the testbed and converted into "clean" datasets. An example of a raw packet and several cleaned packets is shown in Fig. 2.

After the unusable items are removed from the packet, all that remains are the hexadecimal representations of the binary encoded data from the Ethernet transmission. While the data is now cleaned, it is still not ready for input into the various neural networks. Ground truth labels for each sample must be established and final formatting must be completed. The ground truth labels are assigned by a parser built for the purpose. Each stage of the architecture has its own. The formatting of the data is model specific.

To begin, each model input layer has a specific number of desired features. The cleaned samples are either truncated to the appropriate size or padded with zeros. For example, the MLP neural networks take in differing packet sizes. This was done in order to take advantage of different feature combinations while keeping training times low. Previous experimentation revealed that feature counts ranging from 125–160 hexadecimal characters (62.5–80 bytes) worked best for network traffic classification tasks. The three MLP model input packet (feature) sizes are 128, 144, and 156 hexadecimal characters. These sizes serve to include the significant packet headers (Ethernet, IPv4, ICMP, IGMP, UDP, and TCP) without requiring the entire packet which can reach 3028 characters or 1514 bytes after capture.

While the MLP models can now begin to process the data, the CNNs cannot do so. This is because the CNN processes 2-D packet arrays that emulate image data. CNNs are included because much of the contemporary data processing techniques have CNNs at their core. Thus, the packets must now be converted to an image style matrix. Like the MLPs, previous experimentation revealed that converting 196 hexadecimal features (98 bytes) into a square matrix of 14 × 14 works well. Each of the CNNs makes use of the same converted


Before: raw capture

```
+---------+---------------+----------+
20:41:06,254,504   ETHER
|0
|ff|ff|ff|ff|ff|ff|50|7b|9d|d3|3f|89|08|06|00|01|08|00|06|04|00|01|50|
7b|9d|d3|3f|89|81|15|19|ec|00|00|00|00|00|00|81|15|19|fe|00|00|00|00|0
0|00|00|00|00|00|00|00|00|00|00|00|00|00|
```

After: cleaned packets

```
001e27002781000c858eca000800450001483abc0000ff11dfcac0a80fc7c0a80f060043004401
ffffffffffff001e270027810800450002400000000000401178ae00000000ffffffff0044004302
ffffffffffff001e270027a30800450002400000000000401178ae00000000ffffffff0044004302
001e270027a3000c858eca000800450001483abe0000ff11dfcbc0a80fc7c0a80f030043004401
ffffffffffff001e270027a30800450002400000000000401178ae00000000ffffffff0044004302
000c858eca003c970e1b81330800450001 48f660000080110000c0a80f0ac0a80fc70044004301
```

Fig. 2 Raw packet vs. Cleaned packets.

14 × 14 packet images. A gray scale representation of a converted packet image is depicted in Fig. 3.

In this case, the image shows a TCP datagram. The beginning of the IPv4 and TCP headers are indicated. Once this is complete, the CNN can now process the data.

When all the processes are completed, there are four dataset collections: one for each of the three MLP configurations and one for the CNN configuration. The CNN models use the same feature sets but like the MLP models, differ internally.

Another important part of the preprocessing stage is to create the training datasets. Dataset 0 is the training dataset but it is $9{\times}10^4$ samples are for general training. Included are $4{\times}10^4$ UDP and $1.5{\times}10^4$ TCP packets. During the parsing and labeling, these two categories are copied into other training files so that later stages can utilize them.

Lastly, the entire preprocessing runs whenever the architecture is instantiated. However, it can be run offline.

3.3 Neural networks

The system makes use of both CNNs and MLP neural networks. Both have been shown to be accurate when deployed in packet classification tasks and for a variety of applications. The CNNs are comprised of a pair of convolutional and max pooling layers followed by three fully connected layers. The base structure is shown in Fig. 4.

The CNNs differ in the number of filters and size of the fully-connected layers used internally.

The MLPs are comprised of a variable input layer and a pair hidden layers and like the CNNs, an output layer sized to the number of output classes. This configuration is shown in Fig. 5.

The MLPs differ mainly in the size of the hidden layers. However, all of the neural networks can be configured with varying input sizes.

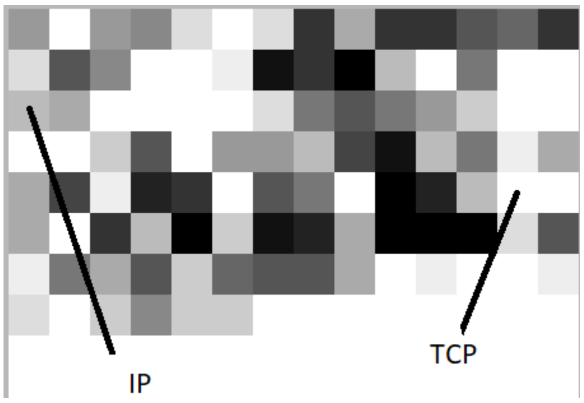


Fig. 3 Packet image.

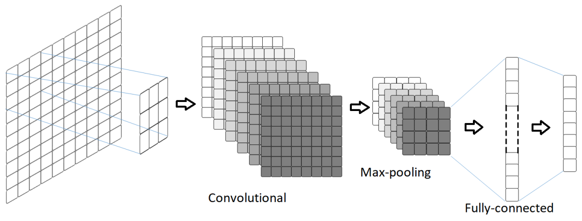


Fig. 4 Base structure of convolutional neural network.

4 Operation

The architecture is comprised of a command module that controls communication between the various components; six neural networks (three CNNs and three MLPs), a series of category specific (general, UDP, and TCP) parsers, recommenders, and a single splitter. The first is represented in Fig. 6. Raw capture packets are supplied to the general wireshark parser which creates the training datasets (features and labels) at all stages and the general test datasets used previeusly.

The recommender completes the task of evaluating all of the results and accuracy values from each of the

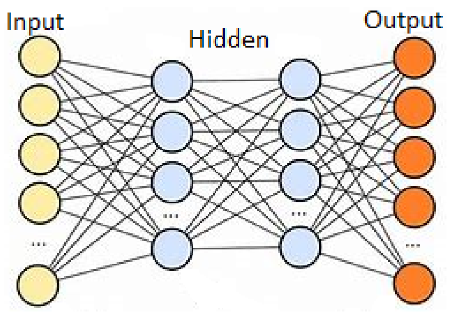


Fig. 5 MLP neural network.

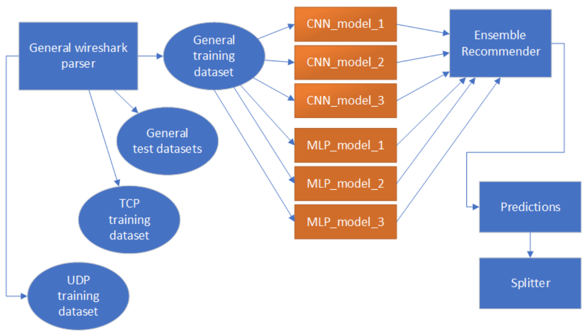


Fig. 6 General stage.

six models. Based on the voting from the recommender, final predictions are made for the general classes. An important last step in this stage is for the splitter to separate of the test datasets into the main categories used later on. For this work, the UDP and TCP samples are separated and new test datasets created. These samples are a subset of the general samples. For example, dataset 6 has 18 610 samples at the beginning of general classification. If the early stage classification was 100% accurate, 14 357 TCP packets would be placed in TCP dataset 6, and 1522 UDP packets would be placed in UDP dataset 6. Thus, it is critical for the accuracy of the general stage to be as accurate as possible.

The next stage to run is for the UDP traffic. This portion of the architecture is shown in Fig. 7.

The UDP training dataset was already created and now the models are trained using this dataset. A goal of this work was to create reusable code, so the models trained are the same ones used earlier as in the recommender.

The parser seen here makes use of the cleaned datasets from the general parser but the labels and features must be recreated because the initial classes were not solely UDP. In addition, it is possible to change the desired features between stages. Once again the recommender evaluates the output from all six models and determines the correct predictions for the results. The UDP process is repeated by the TCP stage.

5 Voting methodology

The whole point of an ensemble is to present a better answer or solution to a problem, typically in the area of classification. In this case, classification accuracy is a precursor to applications that will make use of the class categories downstream which places further weight on the importance on the labeling process. Ensembles can make use of several different voting strategies in order

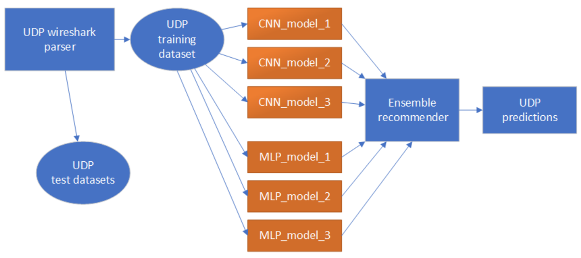


Fig. 7 UDP stage.

to reach the best possible answer. For this work, several are tested and compared.

The first technique is a simple majority vote. Each model calculates the correct label for each sample and the ensemble models compare these labels for the final decision. As the recommender function processes the prediction results for each model, predictions are placed in a vector: $P = [p_0, p_1, \ldots, p_n]$. The list is then evaluated for the value with the greatest number of occurrences giving the updated prediction. This code snippet is seen in Eq. (1).

$$\text{pred} = \text{int}(\max(\text{set}(P), \text{key} = \text{P.count})) \tag{1}$$

This approach met with success for the task of traffic classification; however, as described later, this runs into the problem of a split vote because there was an even number of models. Using the *argmax* function creates an indeterminate result in the event of a tie. For this reason, the lowest performing model is eliminated from the final comparison and the *argmax* function is run on the remaining three models. In addition, models occasionally experience instability with a given model configuration or a particular dataset. This instability results in model performance well outside of the expected accuracy. Eliminating the lowest performer helps address these occasional instability issues.

A simple majority vote is equivalent to applying a weight of one to each vote. Thus, a next step is to modify the weights of each vote based on some criteria, which is often the prior performance of the individual models. Each of the predictions made by ensemble models must be summed, and then a maximum is determined. To this end a vector of weights $W = [w_0, w_1, \ldots, w_n]$ is used to calculate a vector populated with these vote products.

$$\text{Vote} = v_i = \sum_{i=0}^{n} v_i w_i \tag{2}$$

The index of the maximum value in the vector containing the vote summations $V = [v_0, v_1, \ldots, v_n]$ is the final vote for that particular sample.

$$\text{Finalvote} = \max(V) \tag{3}$$

Two weight values are used for these ensembles. The first is a weight based on the validation set accuracy. The justification is that this set contains all of the possible classes for a particular stage. A second weighting strategy based on the overall performance of each model is also explored. This approach offers a broader, average view of a model's behavior.

The majority and the weighted voting strategies are carried forward when the ensemble is converted from four models to six.

6 Ensemble tuning

Tuning a single model can be a time consuming task simply due to the number of variables. Not only can the number of desired input features vary, but the structure and optimizers can change. For example, MLPs are typically described by the number of layers and the width of those layers. Thus, the three MLPs built for this ensemble differ in their input sizes (128, 144, and 156) and in the number of hidden nodes (MLP 1 = 20, MLP 2 = 30, and MLP 3 = 20). All of them are two layer models. Another difference between the models is the batch size used in min-batch gradient descent. In this case MLP 1 = 128, MLP 2 = 64, and MLP 3 = 128. The choice of optimizer is also an important aspect to consider. Currently the ensemble choice is for Adam, but this decision was only made after considerable testing with other optimizers. The configurations having the best performance are shown in Table 2.

While the CNNs use the same number of input features, they are distinctive in the number of layers used, filter configuration, size of the fully connected layers, and even the type of pooling used. For this work, all of the CNN models have two convolutional layers, three fully connected layers, and a max pooling. They differ in the number of filters and the size of the fully connected layers. An example of these are summarized in Table 3.

Table 2 MLP model configuration.

Input	Layers	Hidden node	Batch size	Optimizer
128	2	20	128	Adam
144	2	30	64	Adam
156	2	20	128	Adam

Table 3 CNN model configuration.

Input	Filter 1	Filter 2	Fully-connected layers	Optimizer
196	8	20	80-40-20	AdaDelta
196	12	24	96-48-24	AdaDelta
196	8	20	80-40-20	Adam

Column 4 shows the fully connected layer configurations. All three CNN models use a batch size of 128. It should be noted that all of the configurations seen in Tables 2 and 3 are final configurations that were judged to perform well enough to move to the next stage of research. Even with this small number of variables, both MLP and CNN models have an exhausting number of combinations that might be tried. As mentioned earlier, these configurations were strongly influenced by previous studies on individual models. However, tuning an ensemble can be quite different.

As will be shown later in this article, a model can behave and perform differently depending on the task. All of these variable might be re-tuned for the next stage. For example, learning rates started at 1e-6 and ended up at 1e-3. But later stages were found to have improved performance with a learning rate of 1e-4. What is more, a model performs well at a particular stage, but its performance decreases at a later stage.

Lastly there is the important question regarding the number and type of models to be included in the ensemble. As the next section will indicate, the original ensemble started with four models (2 MLPs and 2 CNNs) and this was later modified to include two more. The type of voting used can also make a difference. In the end, the goal was to achieve high accuracy rates (greater than 99%) in the first stage and reduce training times. After this is accomplished, the model can be further tuned to desired accuracy rates.

7 General result

Initially there were four models used in the ensemble: two CNNs an two MLPs. As stated previously, the models had different architectures with changes made to the structure, the number of filters (CNNs), and the optimizers. The escape loss condition was 1e-6 and the initial learning rate was 1e-6 as well. All of the models used the Adam optimizer. Initially the performance was determined by simple majority vote. Recall that after several hundred iterations, each of the individual models has an accuracy of 98%–99%. If three of the four models determine that a particular sample belongs to a general class based on the greatest probability, the sample is assigned to that class. Experiments were run over 100, 200, 400, 500, 800, and 1000 iterations and the results from all seven datasets are compared. Table 4 contains the results from a 200-iteration experiment.

Table 4 serves to illustrate a number of problems with simple majority voting. The first is that a poorly performing model can bring the performance of the entire ensemble down. This can be seen in two ways. The first is that the average performance of the ensemble lags behind two of the models. In addition, the best performing model is not predictable making it difficult to determine a single model to use for all datasets. Only in the case of dataset 3 did the ensemble succeed. This behavior is consistent regardless of the number of iterations.

Another problem encountered was not as obvious as the reduced performance. The models and the supporting architecture are all written in Python. The class with the greatest number of occurrences (max) for each sample determines the final label. However, what is to be done when two classes have an equal number of votes? The Python 3 documentation states that in the event of a tie, the max function selects the first one encountered. This may not be the correct class for the sample making this an issue for any research project dependent on the decision.

7.1 Model removal

There are essentially three goals at this point: to improve the performance of individual models, to improve the ensemble performance per data set and most importantly, and to improve overall ensemble performance. To this end, the lowest accuracy model was removed from the voting. In this way, the accuracy is improved and there is no longer any need to contend with the possibly uncertain output of the max function. Table 5 depicts the impact of the model eliminating ensemble. For ease of comparison, this is also for 200 iterations.

There are several details that draw attention as the data is analyzed. The first is that the ensemble has either the highest accuracy per dataset or is within 5e-4 of the highest. More importantly is that on average, the ensemble is the best performer (99.42%) with only a single MLP model approaching the same success. Clearly, the elimination of the worst performer is an effective technique in the classification of data packets. Another detail is less obvious, i.e., the CNN models regularly under-perform the MLP models with this low number of iterations.

Table 6 shows the same architecture but after 1000 iterations. As can be seen by the average performance numbers, the ensemble not only reaches the level of accuracy but also increases to 99.6% accuracy.

The CNNs show slight improvements and occasionally attain the highest accuracy. However, the accuracy can be improved and the training time averages 1.5 hours for the two CNN models while the average training time of the MLPs is 13 minutes.

Table 4 Early model comparison.

Data	CNN 1	CNN 2	MLP 1	MLP 2	Ensemble
0	0.9949	0.9976	1.0000	1.0000	0.9984
1	0.9610	0.9759	0.9798	0.9983	0.9734
2	0.9862	0.9898	0.9892	0.9959	0.9913
3	0.9838	0.9835	0.9782	0.9976	0.9974
4	0.9882	0.9865	0.9923	0.9983	0.9956
5	0.9744	0.9689	0.9941	0.9896	0.9687
6	0.9459	0.9711	0.9894	0.9983	0.9846
Ave.	0.9763	0.9819	0.9890	0.9969	0.9871

Table 5 Elimination model comparison.

Data	CNN 1	CNN 2	MLP 1	MLP 2	Ensemble
0	0.9982	0.9977	1.0000	1.0000	1.0000
1	0.9857	0.974	0.9883	0.9911	0.9926
2	0.9923	0.9867	0.9915	0.9926	0.9931
3	0.9685	0.9818	0.9822	0.9911	0.9911
4	0.9772	0.9901	0.9942	0.9958	0.9953
5	0.9929	0.9728	0.9931	0.9925	0.9930
6	0.9909	0.9684	0.9912	0.9934	0.9945
Ave.	0.9865	0.9816	0.9915	0.9938	0.9942

Table 6 Elimination model—1000 iterations.

Data	CNN 1	CNN 2	MLP 1	MLP 2	Ensemble
0	0.9998	0.9999	1.0000	1.0000	1.0000
1	0.9798	0.9774	0.9983	0.9989	0.9989
2	0.9894	0.994	0.9938	0.9896	0.9948
3	0.9801	0.9608	0.9878	0.9966	0.9981
4	0.9908	0.9941	0.9925	0.9958	0.9963
5	0.9940	0.9858	0.9922	0.9926	0.9933
6	0.9703	0.9710	0.9909	0.9983	0.9909
Ave.	0.9863	0.9833	0.9936	0.9960	0.9960

7.2 Improving performance

Improving the performance of the individual models will make them more competitive with each other and thus improve the overall performance of the ensemble. For example, by increasing the number of filters at the convolutional layers in the two models to 12 and 16, respectively, the increase in accuracy is demonstrable. Table 7 reports these results for 1000 iterations.

What can be seen is that modifying the models can result in not only an increase in classification identification rates for each model but also an ensemble recognition rate of 99.77%. However, the cost in training is considerable with the total time for the two CNNs exceeds 5 hours. In addition, the architecture will eventually run two more stages.

7.3 Adding models

What can be done to both improve performance and reduce training time? One choice is to simply add more models so that there are a higher number of votes cast for each sample. However, the additional models must have a corresponding increase in training time. The results from a 500-iteration test are shown in Table 8.

Table 7 Modified models—1000 iterations.

Data	CNN 1	CNN 2	MLP 1	MLP 2	Ensemble
0	0.9998	0.9999	1.0000	1.0000	1.0000
1	0.9864	0.9911	0.9989	0.9981	0.9994
2	0.9973	0.9975	0.9954	0.9966	0.9969
3	0.9749	0.9972	0.995	0.9942	0.9974
4	0.9959	0.9951	0.9958	0.9963	0.9974
5	0.9916	0.9991	0.9914	0.9959	0.9935
6	0.9806	0.9979	0.9982	0.9983	0.9995
Ave.	0.9895	0.9968	0.9964	0.9971	0.9977

Table 8 Adding a model—500 iterations.

Data	CNN			MLP			Ensemble
	1	2	3	1	2	3	
0	0.9989	0.9998	0.9993	1.0000	1.0000	1.0000	1.0000
1	0.9800	0.9900	0.9887	0.9918	0.9886	0.9978	0.9913
2	0.9848	0.9971	0.9933	0.9955	0.9952	0.9952	0.9965
3	0.9915	0.9871	0.9697	0.9925	0.9975	0.9981	0.9969
4	0.9929	0.9929	0.9953	0.9988	0.9988	0.9986	0.9993
5	0.9935	0.9967	0.9935	0.9949	0.9952	0.9967	0.9959
6	0.9587	0.9967	0.9852	0.998	0.9969	0.9958	0.9991
Ave.	0.9858	0.9943	0.9893	0.9959	0.9960	0.9975	0.9970

What is immediately apparent is that all of the models perform reasonably well and that the average ensemble performance matches that of the previous results but after only 500 iterations. These results do come with the anticipated cost of 5 hours in training for all 6 models.

Some of the best techniques to reduce training time are to examine optimizers and learning rates. To this end, learning rates ranging from 1e-3 to 1e-6 were tested along with changes to the optimizers and model structural changes. For space, only the final configuration results are shown in Table 9.

A few additional details are included in Table 9. Datasets 7 and 8 are two datasets comprised of Real-time Transport Protocol (RTP) traffic only. It is clear that the models easily determine the correct classes. It is also clear that the ensemble boasts an average accuracy of 99.91%. This includes the results from the RTP datasets. For comparison, if the models in Table 8 had similar values, the accuracy would increase to 99.73%. It would appear that MLP 2 outperforms the ensemble. This is only true in this particular run. It is often the case that a model may compete with an ensemble. Overall, the average of the ensemble is high.

Table 9 also includes the number of iterations run before reaching the escape loss value of 1e-8 which is a more stringent target. Thus, these results show a significant improvement over simply adding additional models to the ensemble, and total training time is just under 8 minutes as compared to the previous time of 5 hours.

Table 9 Final ensemble.

Dataset	CNN			MLP			Ensemble
	1	2	3	1	2	3	
0	0.9999	0.9999	0.9999	1.0000	1.0000	1.0000	1.0000
1	0.9858	0.9854	0.9954	0.9971	0.9992	0.9981	0.9991
2	0.9960	0.9968	0.9963	0.9918	0.999	0.9946	0.9974
3	0.9992	0.9992	0.9978	0.999	0.9998	0.9971	0.9998
4	0.9924	0.9955	0.9962	0.9989	0.9986	0.9983	0.9989
5	0.9938	0.9943	0.9967	0.9947	0.9992	0.9949	0.9976
6	0.9801	0.9803	0.9817	0.9969	0.998	0.9972	0.9990
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Ave.	0.9941	0.9946	0.9960	0.9976	0.9993	0.9978	0.9991
Iteration	8	7	11	23	14	19	–

8 TCP and UDP results

This section presents the results for the subsequent UDP and TCP stages achieved after the general stage is run and after the results shown in Table 9 are received.

The UDP results in Table 10 are among the most interesting for a number of reasons. The first is that the ensemble is able to correctly distinguish between packets of the same type. There are four different RTP streams and the ensemble (and the models) does not have any difficulty in identification. Second, the models often do not perform as well as they did in the previous stage. For example, MLP 2 was the best performer in the general stage but in the same run, it is among the worst in this stage. Third, some of the models have abysmal accuracy for a particular dataset but the model dropout and voting still allow the ensemble to perform above 98% with about 10 minutes of training time. The TCP ensemble results are shown in Table 11.

Once again, we see good results for very little training time. The TCP trainer is smaller, so the total time is 5 minutes. Like the UDP ensemble, several of the models have poor performance, however, the model dropout and voting keep the ensemble accuracy above 94%. Datasets 7 and 8 do not have values because they are comprised entirely of RTP UDP traffic.

Because this work has additional goals, it was decided that the remainder of the ensemble tuning could be part of future work. However, a reasonable concern might be whether or not the ensemble truly outperforms the various models at all stages. Table 12 provides an "average of averages" for overall performance.

Table 10 Final UDP ensemble.

Dataset	CNN			MLP			Ensemble
	1	2	3	1	2	3	
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	0.9872	0.9587	0.9900	0.9221	0.8945	0.8895	0.9890
2	0.9935	0.9581	0.9852	0.5247	0.5467	0.5443	0.9808
3	0.9758	0.9931	1.0000	0.7232	0.8131	0.7889	0.9931
4	0.989	0.9217	0.9871	0.4742	0.5074	0.419	0.9494
5	0.9985	0.9766	0.9968	0.8894	0.8747	0.8794	0.9895
6	0.9927	0.9350	0.9861	0.6350	0.4957	0.4904	0.9569
7	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Ave.	0.9930	0.9715	0.9939	0.7965	0.7925	0.7791	0.9843
Iteration	22	41	34	65	25	62	–

Table 11 Final TCP ensemble.

Dataset	CNN			MLP			Ensemble
	1	2	3	1	2	3	
0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	0.7580	0.7321	0.8210	0.8972	0.9058	0.8893	0.9251
2	0.8033	0.6988	0.8692	0.8431	0.8954	0.8979	0.9218
3	0.9084	0.9327	0.9402	0.9718	0.9821	0.9764	0.9861
4	0.7510	0.5159	0.6403	0.8686	0.8625	0.6403	0.8695
5	0.3051	0.9322	0.9492	1.0000	0.7797	1.0000	1.0000
6	0.5811	0.6349	0.5769	0.9101	0.9320	0.8379	0.9041
7	–	–	–	–	–	–	–
8	–	–	–	–	–	–	–
Ave.	0.7296	0.7781	0.8281	0.9273	0.9082	0.8917	0.9438
Iteration	22	11	8	83	53	117	–

Table 12 Overall performance.

Stage	CNN			MLP			Ensemble
	1	2	3	1	2	3	
Gen	0.9941	0.9946	0.9960	0.9976	0.9993	0.9978	0.9991
UDP	0.9930	0.9715	0.9939	0.7965	0.7925	0.7791	0.9843
TCP	0.7296	0.7781	0.8281	0.9273	0.9082	0.8917	0.9438
Ave.	0.9056	0.9147	0.9393	0.9071	0.9000	0.8895	0.9757

It is challenging to develop a model that performs well for all applications and in all conditions. These results show that an ensemble constructed in this way can reach high recognition rates for a various of packet traffic type and conditions.

9 Weighted voting

In this section, two different approaches to weighted voting are explored. The first is a strategy based on the performance seen on the validation set. This is done to ensure adequate coverage for the various classes while attempting to minimize the risk of over-fitting. The second strategy looks at overall performance of the models across the validation set and the other test sets. In each case, an update cycle is allowed to run three times with the weights updated at each iteration. The weights are an average of the results seen up to that point. The initial weights for both experiments come from the best performance after tuning the ensemble.

Table 13 provides the initial weights from the validation set results. These values correspond to the performance of various models in the validation set in

Table 13 Initial validation based weights.

Stage	CNN			MLP		
	1	2	3	1	2	3
General	0.9858	0.9854	0.9954	0.9971	0.9992	0.9981
UDP	0.9872	0.9587	0.9900	0.9221	0.8945	0.8895
TCP	0.7580	0.7321	0.8210	0.8972	0.9058	0.8893

Table 16 Initial model average based weights.

Stage	CNN			MLP		
	1	2	3	1	2	3
General	0.9912	0.9919	0.994	0.9964	0.9989	0.9967
UDP	0.9894	0.9572	0.9908	0.6947	0.6886	0.6685
TCP	0.6844	0.7411	0.7994	0.9151	0.8929	0.8736

the highlight performing run but not an indication of ensemble performance.

Table 14 provides the final weights used in the validation series. These are the average of three updates and show the result of the validation set performance.

The changes seen in these weights are small with only CNN 2 changing by more than 0.05%. What is of interest is the impact of the weights on the ensemble performance. Table 15 shows what happens after several weight updates.

Over time, the performance of the general stage does not change very much with the range between the highest and lowest accuracy being 0.0012% or 0.12%. However, the UDP and TCP performances decrease by 4.59% and 1.76%, respectively. This is not a consistent decline as can be seen in Cycle 2. A closer look at the model performance also shows that certain datasets are more difficult for some models to process. It may also be that a greater number of update cycles is needed as Tables 13 and 14 indicate that the weights do not vary greatly from one step to another. For this reason, the basis of weights on average performance was performed.

Table 16 contains the starting weights used during this phase of testing. In addition to a different approach, the values for the training set results and the results for datasets 7 and 8 are not included in the average performance calculation. This was done because the training performance is simply a verification check on model operation and should not create problems for the system. Datasets 7 and 8 also do not create problems having recognition rates of 100% and skew the weights higher. Clearly there is a significant difference between the weights used at this point and the weights based on the validation set with the UDP and TCP values are much lower.

Table 17 provides the final weights used in the validation series. These are the average of three updates and show the result of the average dataset performance on the weights.

Table 18 depicts the results from the various cycles after updating the weights based on model average performance.

The results shown in Tables 15 and 18 are for the ensemble as a whole. In both cases, the general classification task consistently performs well. The strategy using only the weights from the validation dataset does not provide the hoped for results though more update rounds might be needed. The strategy employing weights derived from average performance of several datasets shows a greater potential as the UDP accuracy reaches 99%.

There is another conclusion that can be drawn after

Table 14 Final validation based weights.

Stage	CNN			MLP		
	1	2	3	1	2	3
General	0.9900	0.9744	0.9918	0.9966	0.9976	0.9946
UDP	0.9824	0.9691	0.9718	0.9071	0.8872	0.9008
TCP	0.7647	0.8125	0.7971	0.8972	0.8938	0.8892

Table 17 Final model average based weights.

Stage	CNN			MLP		
	1	2	3	1	2	3
General	0.9935	0.9929	0.9936	0.9965	0.9970	0.9967
UDP	0.9516	0.9478	0.9602	0.6693	0.6815	0.7156
TCP	0.7396	0.7556	0.7162	0.9100	0.8959	0.8866

Table 15 Validation weight performance.

Stage	Initial	Cycle 1	Cycle 2	Cycle 3	Final
General	0.9991	0.9991	0.9979	0.9979	0.9987
UDP	0.9843	0.9464	0.9785	0.9508	0.9384
TCP	0.9438	0.9214	0.9172	0.9391	0.9262

Table 18 Ensemble average weight performance.

Stage	Initial	Cycle 1	Cycle 2	Cycle 3	Final
General	0.9991	0.9986	0.9991	0.9982	0.9979
UDP	0.9843	0.9360	0.9722	0.9904	0.9717
TCP	0.9438	0.9281	0.9331	0.9298	0.9246

the viewing the final results from each model; there is a marked difference in performance between model types. In other words, different model types behave differently at the various stages. All of the models perform well in the general stage. However, the CNNs outperform the MLPs in the UDP classification task and the opposite is true for TCP. Table 19 provides the individual model performance using the average accuracy weight strategy.

Clearly all of the models perform well early on. Results that appear low when compared to the others show the impact of a single dataset rather than reduced overall accuracy. For example, CNN 2 has an average general accuracy of 91.91% but this is due to an error in processing dataset 5 at 39.99% which the ensemble deals with through voting. Without this problem, the CNN 2 average accuracy is 98.4%.

The evidence for the previous conclusion can be seen at the UDP and TCP stages. The average accuracies of the CNNs at the UDP and TCP stages are 96.3% and 78.7%, respectively; while the average accuracies of the MLPs at the UDP and TCP stages are 80.7% and 90.2%. The performance of CNNs is 15% better than that of the MLPs at the UDP stage but nearly 12% worse than that of MLPs at the TCP stage.

As a last step towards improving ensemble accuracy at all stages, two under performing MLP models are removed from the UDP stage and two under performing CNNs are removed from the TCP stage. To be clear, there are now six models in the general stage and four at the UDP and TCP stages. When this is done, the ensemble UDP performance increases to 99.02% compared to a previous high of 98.43% and the TCP accuracy is largely unchanged from its previous high of 94%.

10 Adversarial attack

Network traffic can be manipulated in a variety of ways depending on the nature of the attack. In order to determine ensemble performance in the presence of packet modifications or an attack on the ensemble itself, the incoming data are modified in the following ways:

- Four bytes beyond the IP header are set to 0;
- Four bytes beyond the IP header are set to 9;
- The layer 2 type is changed to 0x0806 indicating ARP traffic; and
- The layer 3 proto ID is changed to 6 indicating TCP.

Dataset 5 is chosen as the sample collection to be poisoned. This is because it does not contain ARP or TCP traffic. It contains a variety of UDP packet types and other layer 2 types as well. The impact of setting the upper layer header to 0 is to remove identifying characteristics which should be problematic for UDP. Setting the header to different values would falsify the header information including the port numbers. Changing the layer 2 types would indicate to a rule-based parser that the entire dataset was made up of ARP traffic. Changing the layer 3 protocol ID to 6 would indicate that the entire dataset was TCP. Importantly, the ground truth for dataset 5 has already been established and is used to the poisoned datasets. Any traditional parser viewing packets with these changes made would either be unable to determine the port numbers causing a packet to be dropped or misinterpret the packet believing it to be ARP or TCP traffic.

The packet distribution for dataset 5 is provided in Tables 20 and 21. The dataset contains a total of 24 354 packet samples. Though this dataset is clearly unbalanced, training for all models is completed with balanced datasets.

After poisoning, the datasets are run through the ensemble in the same way as the non-poisoned datasets. While there is a slight reduction in accuracy in the last scenario (down to 98.98%), the ensemble effectively identifies the packets after data manipulation with accuracies of 99.98%, 99.89%, 99.69%, and 98.98%.

Table 19 Actual model Ave Wt performance.

Stage	CNN			MLP		
	1	2	3	1	2	3
General	0.9987	0.9191	0.9932	0.9974	0.9957	0.9976
UDP	0.9347	0.9877	0.9664	0.7175	0.8311	0.8729
TCP	0.7863	0.8016	0.7726	0.9215	0.9098	0.8757

Table 20 Dataset 5—Layer 2.

Loop	STP	CDP
2907	14 534	484

Table 21 Dataset 5—Layers 3 and 4.

IGMP	DNS	DHCP	SSDP	NBNS
2356	868	67	3111	27

The distribution results of the four scenarios are shown in Table 22. Dataset 5 performance is included for comparison. For clarity, only the impacted classes are shown.

The descriptions are necessarily shortened but refer to a wiping of the header data, modification of header data, changing the L2 type to ARP, and changing the L3 protocol ID to TCP. The TCP column is included to show the impact of the protocol ID change. The ensemble general and UDP accuracy are included in the last two columns. As can be seen, the ensemble is not fooled by the packet modifications though a traditional parser would.

Comparing the individual models to the ensemble reveals that while a model occasionally out performs the ensemble, this is not a consistent result. Over three consecutive trials, the ensemble accuracy for all four poisoning attacks was 97.5% with the outliers included. The best performance for a single model was 92.27%. The ensemble performs above 99.9% on UDP header manipulation and layer 3 protocol ID field changes. In these three cases, one or two of the models achieve similar results, though not the same ones. The layer 2 manipulation is more challenging causing several models to drop below 40% though the ensemble achieves 91.5%. In this case, two of the models achieve greater than 99% making this attack part of future research.

11 Discussion

This article began with the construction of an ensemble comprised of four neural networks: two CNNs and two MLPs. Each model has a unique configuration. While the ensemble works reasonably well, achieving better than 89% accuracy in the general packet classification task, training time is longer than desired. In addition, the ensemble will eventually be working in a series of stages with the later stages depending on the accuracy early on. Thus, it was also important to increase accuracy while reducing training time. Here, several strategies are considered, including modification of the ensemble structure, tuning of the ensemble, and different voting methods.

Table 22 Dataset 5—Poisoned.

Test	IGMP	DNS	NBNS	TCP	General accuracy	UDP accuracy
DS 5	2356	738	90	67	0.9972	0.9843
zero'd	2358	800	91	2	0.9998	0.9843
9's	2356	780	88	27	0.9989	0.9849
ARP	2357	731	89	75	0.9969	0.9845
TCP	2356	579	68	248	0.9898	0.9893

The two changes made at early stages were the increase of the ensemble size to include six models (adding a CNN and an MLP) and then a model drop out in which the lowest performing model is simply removed. This also alleviates the problem of voting that utilizes an *argmax* function decision because a tie is less likely at these high model accuracies. At this point, the ensemble models take a simple majority vote on the best class for each sample. These strategies are effective and resulted in an accuracy increase to 99.7% in the general classification task. However, this accuracy comes at a cost of 5 hours of ensemble training time.

11.1 Tuning

The next step was to tune the various models and implement the successive stages. In tuning the models, the best optimizer, learning rates, layers, filter configuration (CNN), and hidden nodes (MLP) are explored. In the end, the Adam and AdaGrad optimizers are used along with the structures shown in Tables 2 and 3. Learning rates of 1e-3 and 1e-4 are used instead of the previous 1e-6. This process has the effect of achieving a performance of 99.91% with 10% of the training time.

The general stage is the first of three stages. The entire ensemble is run again in order to process the separated UDP packets and then a third time is performed to process TCP packets. While some tuning is done between stages (iterations, learning rates), the models are not changed in order to be able to compare results. With majority voting, model dropout and tuning general ensemble, UDP, and TCP stages have performance averages 99.9%, 98%, and 94%, respectively. These averages become the target values to exceed in the subsequent voting methodology and investigations.

11.2 Voting

There are two weighting strategies investigated. The first is based on the performance each model achieves on the validation set. The second is on each model's overall performance on the challenging datasets. In the first case, the results are inconclusive. The initial weights are from a high-performing configuration and the weights change significantly. The results of the model average performance are more encouraging. In both cases, the models achieve 99% for the general stage. Average weighting reaches this for the UDP stage as well. TCP performance is largely unchanged for both strategies.

Of interest is the behavior of different model types during the various stages. CNNs perform better for UDP and MLPs are better at TCP recognition. The reasons for these results are unclear however it may be that the model types "visualize" the data differently and certain features stand out more. In any case, this led to the later stage ensemble modification to operation: Two MLPs were removed from the UDP processing and two CNNs were removed from TCP processing. The result was an increase in UDP accuracy to above 99%. TCP performance must still be tuned.

It is tempting to simply use CNN models for UDP processing and MLP models for TCP. However, this approach is problematic for several reasons. The first concern is the processing of additional classes. While a pattern seems to emerge from the work done so far, it is not certain that the same behavior will continue as the classes expand. Second, models can be unstable. For example, when processing TCP packets, the MLP model performance varies by more than 10% and the CNN models achieve greater than 95%. The combination of these results and the different method of processing the data increases the performance of the ensemble. Lastly, a different dataset can also cause a model to perform differently. The same variation noted previously occurs with CNNs, sometimes on the same dataset.

It should be noted that the ensemble has limitations. Occasionally, one of the models will have a recognition rate nearly 40% likely because of the fast learning rate of 1e-3. As the models aim for function minimums, the faster learning rate may cause them to "chase" a gradient in the wrong direction or get stuck in a local minimum. Some datasets can cause more problems than others. An example can be found in dataset 4 during the TCP stage. None of the models stand out as regularly being able to reach 90% accuracy. In previous work, this was due to problems in model tuning and class definitions. Often splitting a class into multiple sub-classes adequately addresses the issue.

11.3 Adversaries

Dataset 5 was targeted for poisoning. Four new datasets are created, each with a different strategy for reducing the performance of the ensemble. However, the new datasets use the same labels as the original dataset 5. There are two attempts to destroy or modify upper layer header information and two attempts to change the fields used to identify packet or frame type. After dropout, tuning, and ensemble structure modifications, none of the attacks were successful in dropping the accuracy below 98% and only one attack was able to reduce performance below 99% for either the general or UDP cases (see Table 22). While the TCP stage was not investigated at this point, the dataset was poisoned to make the ensemble believe that the traffic was TCP based. As just mentioned, this attack was not successful.

12 Conclusion

CNN and MLP neural networks have been used for packet classification tasks; however, they suffer from occasional instability and dataset-specific errors. Neural network ensembles such as the one described here use voting strategies and redundant decision making to increase accuracy. This architecture is divided into three stages: general, UDP, and TCP. After training and tuning are performed the general and UDP stages reach 99% accuracy while TCP achieves 94% for a variety of traffic classes and datasets. It was also found that the expected increase in training time is more than compensated by tuning of each individual model.

However, neural network models can be fooled by adversarial examples that seek to create miss-classification errors. Thus the ensemble was tested against a variety of packet modification attack directed at header and content manipulation. In three or four cases, the ensemble kept accuracy above 99%. In the fourth case, it remained above 98% accuracy. This work shows the importance of neural network ensembles not only for very high levels of accuracy but also for defending against these types of attack. The neural network ensemble is more robust than traditional parsers or rule based systems.

References

[1] B. Hartpence and A. Kwasinski, Fast internet packet and flow classification based on artificial neural networks, in *Proc. 2019 SoutheastCon*, Huntsville, AL, USA, 2019, p. 19433953.

[2] B. Hartpence and A. Kwasinski, A convolutional neural network approach to improving network visibility, in *Proc. 2020 29th Wireless and Optical Communications Conf.* (*WOCC*), Newark, NJ, USA, 2020, pp. 1–6.

[3] A. J. C. Sharkey, N. E. Sharkey, U. Gerecke, and G. O. Chandroth, The "test and select" approach to ensemble combination, in *Proc. 1st Int. Workshop on Multiple Classifier Systems*, Cagliari, Italy, 2000, pp. 30–44.

[4] T. G. Dietterich, Ensemble methods in machine learning, in *Proc. 1st Int. Workshop on Multiple Classifier Systems*, Cagliari, Italy, 2000, pp. 1–15.

[5] M. A. Yaman, A. Subasi, and F. Rattay, Comparison of random subspace and voting ensemble machine learning methods for face recognition, *Symmetry*, vol. 10, no. 11, p. 651, 2018.

[6] U. Knauer, C. S. Von Rekowski, M. Stecklina, T. Krokotsch, T. P. Minh, V. Hauffe, D. Kilias, I. Ehrhardt, H. Sagischewski, S. Chmara, et al., Tree species classification based on hybrid ensembles of a convolutional neural network (CNN) and random forest classifiers, *Remote Sens.*, vol. 11, no. 23, p. 2788, 2019.

[7] E. Tasci, Voting combinations-based ensemble of fine-tuned convolutional neural networks for food image recognition, *Multimed. Tools Appl.*, vol. 79, no. 41, pp. 30 397–30 418, 2020.

[8] I. E. Livieris, A. Kanavos, V. Tampakas, and P. Pintelas, A weighted voting ensemble self-labeled algorithm for the detection of lung abnormalities from x-rays, *Algorithms*, vol. 12, no. 3, p. 64, 2019.

[9] T. Strauss, M. Hanselmann, A. Junginger, and H. Ulmer, Ensemble methods as a defense to adversarial perturbations against deep neural networks, arXiv preprint arXiv: 1709.03423, 2017.

[10] W. Q. Wei, L. Liu, M. Loper, K. H. Chow, E. Gursoy, S. Truex, and Y. Z. Wu, Cross-layer strategic ensemble defense against adversarial examples, in *Proc. 2020 Int. Conf. Computing, Networking and Communications* (*ICNC*), Big Island, HI, USA, 2020, pp. 456–460.

[11] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, Deep residual learning for image recognition, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, NV, USA, 2016, pp. 770–778.

[12] K. Simonyan and A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv: 1409.1556, 2014.

[13] B. Pan, Z. W. Shi, and X. Xu, Hierarchical guidance filtering-based ensemble classification for hyperspectral images, *IEEE Trans. Geosci. Remote Sens.*, vol. 55, no. 7, pp. 4177–4189, 2017.

[14] S. Gunderson and F. Jagodzinski, Ensemble voting schemes that improve machine learning models for predicting the effects of protein mutations, in *Proc. 2018 ACM Int. Conf. Bioinformatics, Computational Biology, and Health Informatics*, New York, NY, USA, 2018, pp. 211–219.

[15] E. Sin and L. P. Wang, Bitcoin price prediction using ensembles of neural networks, in *Proc. 2017 13th Int. Conf. Natural Computation, Fuzzy Systems and Knowledge Discovery* (*ICNCFSKD*), Guilin, China, 2017, pp. 666–671.

[16] L. Y. Xu, X. Zhou, Y. M. Ren, and Y. F. Qin, A traffic classification method based on packet transport layer payload by ensemble learning, in *Proc. 2019 IEEE Sympo. Computers and Communications* (*ISCC*), Barcelona, Spain, 2019, pp. 1–6.

[17] S. E. Gómez, L. Hernández-Callejo, B. C. Martínez, and A. J. Sánchez-Esguevillas, Exploratory study on class imbalance and solutions for network traffic classification, *Neurocomputing*, vol. 343, pp. 100–119, 2019.

[18] S. E. Gómez, B. C. Martínez, A. J. Sánchez-Esguevillas, and L. H. Callejo, Ensemble network traffic classification: Algorithm comparison and novel ensemble scheme proposal, *Comput. Networks*, vol. 127, pp. 68–80, 2017.

[19] F. Gargiulo, L. I. Kuncheva, and C. Sansone, Network protocol verification by a classifier selection ensemble, in *Proc. 8th Int. Workshop on Multiple Classifier Systems*, Reykjavik, Iceland, 2009, pp. 314–323.

[20] CAIDA, https://www.caida.org/home/.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, Rethinking the inception architecture for computer vision, in *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition* (*CVPR*), Las Vegas, NV, USA, 2016, pp. 2818–2826.

[22] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, Ensemble adversarial training: Attacks and defenses, arXiv preprint arXiv: 1705.07204, 2017.

[23] M. Nasr, A. Bahramali, and A. Houmansadr, Blind adversarial network perturbations, arXiv preprint arXiv: 2002.06495, 2020.

Andres Kwasinski received the diploma degree in electrical engineering from the Buenos Aires Institute of Technology, Buenos Aires, Argentina, in 1992 and the MS and PhD degrees in electrical and computer engineering from the University of Maryland, College Park, MD, USA, in 2000 and 2004, respectively. He is currently a professor at the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY, USA. He has co-authored over 90 publications in peer-reviewed journals and international conferences. He has also co-authored the books *Cooperative Communications and Networking* (Cambridge University Press, 2009) and *3D Visual Communications* (Wiley, 2013). His current areas of research include cognitive radios and wireless networks, cross-layer techniques in wireless communications, and signal processing applied to smart infrastructures. He is also the chief editor of the *Signal Processing Repository* (SigPort) and an area editor for Special Initiatives of the *IEEE Signal Processing Magazine*.

Bruce Hartpence received the PhD degree in computing and information sciences from the Rochester Institute of Technology in 2020, and the MS degree in information technology from the same institute from in 1998. He is currently a professor for the GCCIS i-School at the Rochester Institute of Technology (RIT) where he teaches networking and communication. For the last several years, he has explored the application of neural networks to communication challenges. He has also authored several books and video series for O'Reilly publishing including the *Packet Guide* series. He is also a co-developer of the IEEE 1910.1 Meshed Tree standard for the IEEE. His current areas of research include neural network ensembles applied wired and wireless, real time communication, and intelligent networking.