5-2020

Neural Network Architectures and Ensembles for Packet Classification: Addressing Visibility, Security and Quality of Service Challenges in Communication Networks

Bruce Hartpence
bhhics@rit.edu

Neural Network Architectures and Ensembles for Packet Classification: Addressing Visibility, Security and Quality of Service Challenges in Communication Networks

by

Bruce Hartpence

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences
Rochester Institute of Technology

May 2020

Signature of the Author ____________________________________

Certified by ____
PhD Program Director   Date

Ph.D. IN COMPUTING AND INFORMATION SCIENCES
ROCHESTER INSTITUTE OF TECHNOLOGY
ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

Ph.D. DEGREE DISSERTATION

The Ph.D. degree dissertation of Bruce Hartpence
has been examined and approved by the
dissertation committee as satisfactory for the
dissertation required for the
Ph.D. degree in Computing and Information Sciences

Dr. Andres Kwasinski, Dissertation Advisor

Dr. Victor Perotti, External Chair

Dr. Minseok Kwon

Dr. Raymond Ptucha

Dr. Shanchieh Yang

Date

DISSERTATION RELEASE PERMISSION
ROCHESTER INSTITUTE OF TECHNOLOGY
GCCIS Ph.D. PROGRAM IN COMPUTING AND INFORMAITON SCIENCES

Title of Dissertation:

Neural Network Architectures and Ensembles for Packet Classification: Addressing Visibility, Security and Quality of Service Challenges in Communication Networks

I, Bruce Hartpence, hereby grant permission to Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Signature ____
Date

Neural Network Architectures and Ensembles for Packet Classification: Addressing Visibility, Security and Quality of Service Challenges in Communication Networks

by

Bruce Hartpence

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences
Ph.D. Program in Computing and Information Sciences
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

Increasingly researchers are turning to machine learning techniques such as artificial neural networks (ANN) to address communication network research challenges in the areas of enhanced security, quality of service, visibility and control. Central to each is the need to classify packets. Determining an effective architecture for the artificial neural network is more difficult because traditional techniques such as principal component analysis (PCA) show reduced effectiveness. Presented are the techniques for preprocessing datasets and selecting input traffic features for the multi-layer perceptron (MLP) architecture. This methodology achieves classification accuracy above 99%.

An investigation into neural network architectures revealed the optimal structure and parameters for communication packet classification. This work also studies optimization algorithms with completely balanced datasets and provides performance criteria for training time and accuracy.

The application of MLPs to security challenges is also investigated. Port scans are a persistent problem on contemporary communication networks. Sequential MLPs are investigated to classify packets and determine TCP packet type.

Following classification, analysis is performed in order to discover scan attempts. Neural networks can be used to successfully classify general packet traffic and more complex TCP classes at rates that are above 99%. The proposed methodology achieves accurate scan detection without having to utilize an intrusion detection system.

In order to harness the power of Convolutional Neural Networks (CNNs), the conversion of packets to images is investigated. Additionally, a sequence of packets are combined into larger images to gain insight into conversations, exchanges, losses and threats. The use of this technique to identify potential latency problems is demonstrated. This approach of using contemporary network traffic and convolutional neural networks has success rate for individual packets exceeding 99%. Larger images achieve the same high level of accuracy. Finally, neural network ensembles are researched that reach 100% accuracy for packet classification.

Ensembles are also studied to accurately predict Mean Opinion Score for voice traffic and explored for their use in combating adversarial attacks against the source data.

Acknowledgments

I would like to express my gratitude to Dr. Andres Kwasinski for his knowledge, guidance, insight and willingness to work with an odd duck. I would also like to thank the members of my dissertation committee; Dr. Shanchieh Yang, Dr. Minseok Kwon and Dr. Raymond Ptucha for adding their expertise and perspective. Lastly, I deeply appreciate the patience and support from the Ph.D. program faculty.

The work that went into this thesis and the supporting course of study consumed many years. For my family it became a way of life and something that they simply accepted. I thank them for the support, the sacrifice of time as I struggled my way through and the final cheer-leading as the finish line got close. For my wife Christina, I would also add appreciation for the occasional gut-check and grounding necessary for endeavors such as this. So, Christina, Brooke, Nicholas and Sydney - this is dedicated to you. And if there are any other old dogs that think it can't be done - sure, its not easy, but you miss every shot you don't take. And what's the worst that could happen?

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Network Research and Machine Learning

Communication networks represent several important areas for study. These include graph theory, connectivity, algorithm design, protocol engineering and many others. The advent of the Internet of Things (IoT), 5G and the increased use of virtualization have created new, or least magnified, problems for modern communication networks. These problems include a very high number of devices, nodes that come and go with increased frequency, demands on capacity, quality of service/experience, security threats and visibility black holes. While there are many aspects to each, of critical importance is an understanding of data packets and packet flows. Today, the ability to apply more powerful algorithms and greater processing power has facilitated the application of machine learning to these network data streams. As a result, machine learning techniques and architectures to address network visibility, quality of service and security have become major focal points for both industry and academia.

However, applying machine learning techniques to network problems is not without its own issues. Researchers experiment in the blind with structures or hyper-parameters without reasoning behind the choices made. It is common to see architectures utilized simply because it was successful for some other application. Thus, available works do not document their rationale for choices made or fail to experiment with varying parameters because they were not part of the problem studied. The central difficulty with this approach is that researchers lack a body of knowledge that can be used as a basis for the work.

As examples, and for the contributions discussed herein, both multi-layer perceptron (MLP) and the Convolution Neural Network (CNN) models have been used to address not only packet classification, but security and performance issues as well. Both of these neural network types have a wide variety of choices to be made with regard to their architecture and operation. These include layers, learning rates, optimization techniques, filter sizes and so on. Very few of the works available today discuss these values and even fewer document the reasons for these values. This potentially opens the field to include exploration that would discover these answers. Examples of MLP and Convolutional Neural Networks are shown in Figure 1.1.

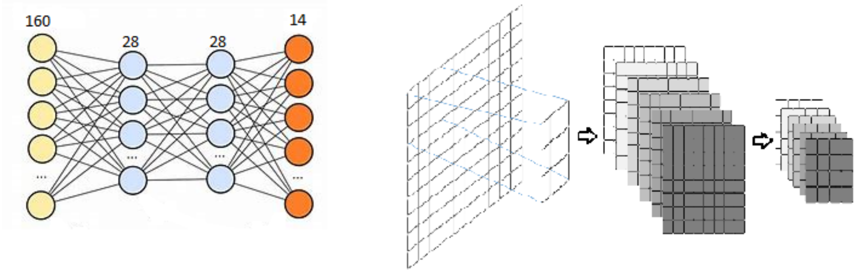


Figure 1.1: MLP and Conv Neural Networks

The motivation for this work comes from the unacceptable performance found in published models for packet classification. While often greater than 90%, this simply will not be sufficient when it comes to building on this accuracy to address greater challenges. Motivation can also be found in the need to address continuing network problems with the greater power of machine learning. An important part of this work is to address the first in order to address the second. The end goal is to investigate a problem such as quality of service knowing that the packet classifier is true. Machine learning models must also be investigated for their effectiveness in a particular application. A critical point is missed when a model is simply accepted as valid. Less obvious sources for motivation come from the lack of access to datasets and the techniques for effectively utilizing these datasets to solve problems.

1.2 Packet Classification Challenges

Traffic classification is central to understanding network operation and conditions. With the creation larger and larger volumes of traffic and data, the problem of rapid and accurate classification becomes more difficult. In addition, packets are often bound together into flows and these flows, rather than individual packets, are often how traffic is moved through a system. For example, Cisco Express Forwarding [66] pre-calculates flow forwarding decisions and Openvswitch [50] utilizes flow tables extensively. Protocols such as Netflow [12] are based on understanding or sending telemetry about these flows. This research work begins with the goal of understanding individual packet classes and then address particular types of port scanning and Voice over IP flows.

An important observation is that if the packet classification problem can be solved, the much larger and more complex issues such as security can be more readily addressed.

Researchers attempting to attack these problems are faced with the need to gain access to datasets but access to the data is often limited. For example, privacy concerns may prevent the release of actual packets, requiring researchers to classify packets or flows from telemetry information alone. But this approach can create a loss of context and is only a window into communication. Protocols such as Netflow are sampling overall traffic and the information provided is a 5-tuple of addressing and message type. This means that not only is the information a subset of actual traffic, the fields provided are a subset as well.

Even if granted access to data, resources typically require analysis and preprocessing. For Internet research, traffic analyzers like tcpdump and Wireshark both save traffic in pcap files but these require specialized software. For other popular datasets [45], the information is stored for use by Weka [23] in tool-specific arff files.

Data age is another consideration. Today the community uses a collection of well-known repositories such as [45] [75] [35] but these are becoming outdated. This older data may no longer represent current network traffic or threats [15]. Other datasets are highly specific so they do not fit into general network research or they are not comprised of actual traffic at all.

To avoid these issues, a local testbed was constructed and contemporary traffic datasets were created. The construction of the datasets is covered in a later chapter. This work also addresses some of the challenges associated with building balanced network datasets.

Several research fields that process complex or high-dimensional data have successfully deployed neural networks. Neural networks provide additional tools necessary to process large datasets in real time while also being able to work on a variety of inputs. The contention here is that they can be leveraged for communication systems. Many researchers believe that the combination of machine learning and visibility are requirements for solving many network problems [5]. Projects such as those described in [77] describe not only circumstances that might benefit but also the datasets that might be extracted using neural networks. Neural networks and software defined networking are combined in works such as [78] and [43]. In both cases, the combination of neural network architectures and software defined networking are pillars of future communication.

For the work described here, neural networks take streams of raw hexadecimal information to determine objective functions that make it possible to identify traffic or flows without additional details. Normally, this hexadecimal series would contain certain features or fields (ex. Ethertype) that a dissector in a program like Wireshark or tcpdump would use to uniquely identify the packet. However, a machine learning model can be trained to identify the same packets without having to first decode field values. The packet analyzer is no longer needed.

The multilayer perceptron models used in the first stages of this work achieve accuracies exceeding 99%. This neural network approach does require a preprocessing/parsing step before the data can be utilized. For this work, four Python based parsers were written to process the variety of available resources.

For the contributions discussed herein, both multi-layer perceptron (MLP) and the Convolution Neural Network (CNN) models have been used to address not only packet classification, but security and performance issues as well. For greater accuracy, neural network ensembles are created. These ensembles can also combat adversarial attacks against the datasets themselves.

Thus, a central theme of this research is to apply neural networks to the data obtained from an operating topology in order to improve packet recognition and use this as a base to address other challenges. This work will be under the umbrella of a framework that takes in network telemetry and provides insight into network conditions and behavior.

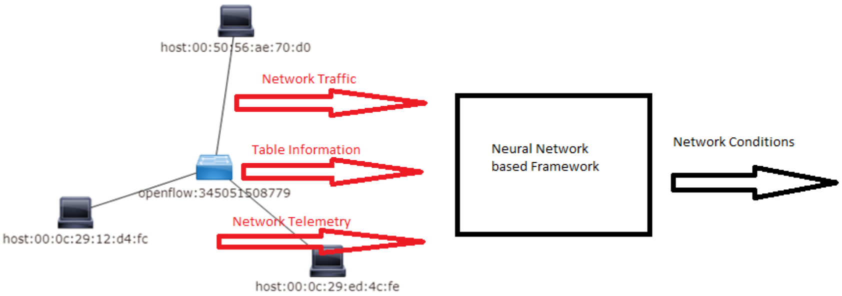


Figure 1.2: Network Visibility Framework

1.3 Optimization

Neural networks can be trained to address a variety of problems and are commonly used in facial recognition, images identification and video stream analysis. Much less is known about their use in tasks specific to communication packet traffic because the area has received less attention. While there are papers addressing neural network packet traffic classification [41] [29] [44], many questions still remain. Researchers still seek the proper type of neural network, structure, hyper-parameter values and most proficient optimization algorithms. It is common to see the same techniques applied to different problems without an understanding of why a choice is being made.

A comprehensive study for the packet classification task is needed because incomplete or incorrect information can lead to excessive training time or model instability. Optimization techniques can reduce this time but researchers have little guidance as to what algorithm to choose and why. In addition, the optimization algorithms (RMSprop, Adam, etc.) each come with their own set of configurable parameters. Thus, an extension to highly accurate packet classification using neural network is to determine the best neural network structure and features to use when dealing with Internet traffic; one that is optimized for accuracy and speed. The research herein explores the limits of algorithm accuracy as a function of regularization, momentum and available helper algorithms.

1.4 Security

Neural networks can be effective at addressing more complicated challenges such as security threats. Attacks against communication networks often start with some form of reconnaissance. One of the best known methods for accomplishing this is the transmission control protocol (TCP) port scans. Attackers look for vulnerabilities of a particular type across a collection of targets or attempt to determine all of the possible attack vectors on a single host. In a TCP port scan, an attacker sends TCP datagrams with the SYN flag set can be sent to each of the 65535 TCP ports on the target(s). The target host will have one of two responses; if a port is "open", the target will respond with a TCP datagram setting the SYN and ACK flags. If a port is "closed", the target will respond with a datagram having the RESET or RESET and ACK flags set.

Unfortunately, readily available tools like NMAP allow even unskilled attackers to perform a variety of scans. Analyzers such as Wireshark provide packet decodes and statistics but leave it to the user to determine if a port scan occurred. Detectors such as Snort are rule based making them static, complex and prone to false positives if incorrectly configured. Proprietary tools can provide greater detail but can be a black box, hiding the techniques and analysis details.

Though they have acknowledged weaknesses, statistical techniques for detecting intrusions or scans have existed for decades [32]. Rule based systems such as Snort are also commonly employed and while the rules start off simple, they become more complex as exceptions arise or an increasing number of situations are addressed. Rules are also predictable and deterministic in that they are matched (or not) which can result in false positives and false negatives. Attackers can take advantage of this predictable behavior. Lastly rule based systems cannot adapt to changes.

Trained machine learning techniques such as neural networks can speed the processing of test datasets and can be very flexible in their ability to accept a variety of inputs and changing network traffic patterns. A neural network ensemble can not only address concerns like this, it may also be used to detect injected data or attempts at poisoning the datasets.

In this work, an ensemble of sequential neural networks was deployed in order to break the complex scan detection task down to its component parts and learn from current conditions. Following neural network processing, a scan detector works with the neural network output to make further determinations about the traffic.

1.5 CNNs

The early research described in this work utilized multi-layer perceptron (MLP) neural networks. However, for many applications (especially image and video processing), MLPs have been superseded by Convolutional Neural Networks (CNNs). Thus, the latter portion of this research utilized either CNNs or a combination of CNNs and MLPs. When compared to MLPs, CNNs process data in a completely different fashion so packets are re-imagined as images. This also allows different and perhaps more contemporary architectures to be investigated so that we can begin to apply techniques that have recently made so many advances.

Using the CNN to classify packets is an important aspect of the work but a separate goal is to apply a novel organization by placing packets into larger images. These images represent a period of time determined by the number of images combined. This larger view can be used to detect network performance issues or attack profiles. Visualization techniques are used in an attempt to more easily understand a system or set of conditions. A system is posited that might use an images such as this to recognize patterns or events in the data and potentially act on them. The trained CNN is now the engine that is applied to each section of the image to determine the packets that are present. Once trained, the CNN is very fast which allows the processing of static datasets or live packet capture.

1.6 Real Time Applications and Data Protection

The classification and port scan ensemble ideas previously presented are highly accurate with recognition rates exceeding 99%. However, even this level of accuracy may be insufficient for tasks such as packet/flow forwarding through a software switch or router. In addition, as the number of classes increases, the error rate may also increase. Sequential neural networks or networks having errors in the early stages, suffer from compounding of these errors. Lastly, the current system is not protected from data poisoning or injection. For these reasons a more powerful ensemble that breaks the identification down to first recognize the various classes to an accuracy of 100% is constructed. The ensemble chooses the best performing model and uses it as the trainer. This approach provides greater confidence in the results at each stage and is the mechanism by which erroneous or malicious data can still be correctly classified. Neural networks and other packet processing constructs such as access control lists or firewalls can be fooled by packet injection or falsification attacks. A neural network ensemble can be utilized to determine whether data has been manipulated in this way.

In addition to the research questions already mentioned, communication networks have also come to depend upon specific real time applications such as Voice Over IP (VoIP). These applications require close monitoring of latency, packet loss and jitter because of their direct impact on performance. For these reasons, machine learning techniques have recently been used to aid in the detection of anomalous traffic and to help determine current conditions [29] [20] [25] [70].

An application of the system can be found in the Mean Opinion Score (MOS) for VoIP. One of the products of the ensemble is the collection of User Datagram Protocol (UDP) packets. These in turn are broken down into the UDP processes including the voice traffic. The voice packets are then run against a Mean Opinion Score (MOS) neural network that takes network conditions and converts them into an MOS score. This achieves another goal of the work in improving visibility into a segment and addressing application performance.

1.7 Contributions

The contributions of this work then include successful neural network architectures for the classification of a variety of contemporary packet types. This investigation answered this question with both multi-layer perceptron and convolutional neural network models. Further, we have determined the proper structures to use for packet classification and as a basis for tackling larger challenges. These structures were determined through exhaustive experimentation that covered the number and size of layers, optimization techniques that work well for communication applications and the hyper-parameters that are effective. In the case of convolutional neural networks, we also provide details on the number of filters used, their stride and size. In several areas we make use of neural network ensembles and sequential networks to solve more complicated problems and ensure accuracy.

Several other contributions relate to the applications that can benefit from the use of neural networks. This includes an investigation into addressing port scanning attacks, visibility and quality of service through a model that predicts Mean Opinion Score.

Behind all of this is the need to understand dataset management. Thus, another important contribution is our investigation into balanced datasets, classes to be classified in a modern communication network and performance aspects of the neural network in terms of training time, accuracy and reliability.

Chapter 2

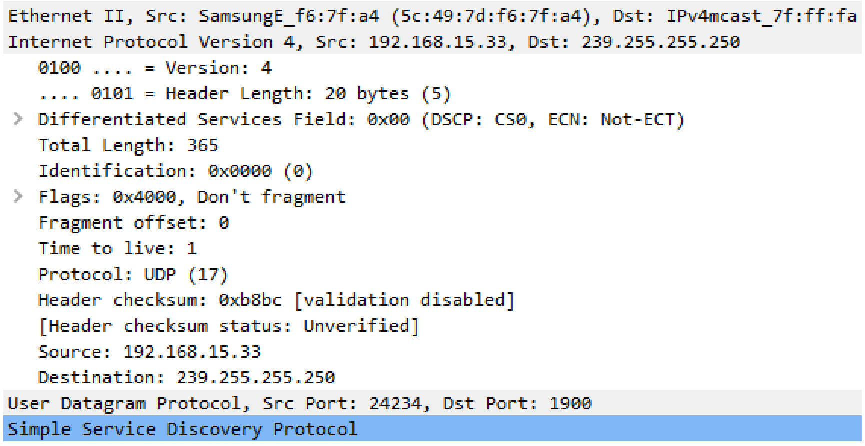
Related Work

2.1 Packet Classification

Examples of this need to understand traffic can be found in [62] and [69] in which neural networks are used to detect anomalies. In both cases the systems must first understand packets and/or flows prior to processing. Notably, some of the datasets used date from 1999. A significant classification effort can be found in [4] in which the authors use flow statistical properties for identification. In a follow up work [44], they modify their approach, applying neural networks to the same dataset after selecting the desired features. The dataset used was created in 2003.

This 2003 dataset collection is described in [45]. The datasets were collected from a particular site over time and are comprised of 249 detailed packet features followed by a label for the class. The authors do not use all packet fields. For example, in [4] the server port is not used for processing, though it is included in the dataset. While the classification rate is high (approx 99%), the authors note that the data is heavily weighted towards a WWW class and so the classification rates for the other classes vary. The authors also limit themselves to complete TCP connections.

An important point made in this paper is that network traffic may change over time and in our work we have found this to be absolutely true. This underscores our desire to work with updated traffic. As part of our work, we emulate some of these results in order to understand the perspective of other researchers. However, we depart significantly after this as our focus turns to the actual packets and flows seen on our testbed. An example of a network packet is shown in Figure 2.1.



```
Ethernet II, Src: SamsungE_f6:7f:a4 (5c:49:7d:f6:7f:a4), Dst: IPv4mcast_7f:ff:fa
Internet Protocol Version 4, Src: 192.168.15.33, Dst: 239.255.255.250
   0100 .... = Version: 4
   .... 0101 = Header Length: 20 bytes (5)
 › Differentiated Services Field: 0x00 (DSCP: CS0, ECN: Not-ECT)
   Total Length: 365
   Identification: 0x0000 (0)
 › Flags: 0x4000, Don't fragment
   Fragment offset: 0
   Time to live: 1
   Protocol: UDP (17)
   Header checksum: 0xb8bc [validation disabled]
   [Header checksum status: Unverified]
   Source: 192.168.15.33
   Destination: 239.255.255.250
User Datagram Protocol, Src Port: 24234, Dst Port: 1900
Simple Service Discovery Protocol
```

Figure 2.1: Network Packet

Another flow classification effort using the RedIRIS datasets can be found in [41]. In this work the authors use complex recurrent and convolutional neural networks to classify with 108 different labels using only six features (source port, destination port, payload bytes, TCP window size (set to 0 for UDP), inter-arrival time, packet direction). The ground truth labels were not provided but had to be derived from analysis using nDPI. The authors report that their best accuracy of 96.32% is achieved with the simplest model for aggregated classes but they achieved high results in the top fifteen most frequent labels using "one vs. rest" classification. For comparison, we regularly achieve recognition rates of 98-99%.

But persistent problems exist in a majority of these works; older datasets and even if the structure of the neural networks is documented, the reasons for the structure or parameter choices are not.

2.2 Datasets

In the study of neural network classification of packet traffic, structure of the datasets used during training, validation and testing must be determined. The typical workflow for machine learning research involves the use of packaged libraries (e.g. pytorch and python, Kera, tensorflow) which includes the optimizers and dataloaders. Researchers often begin with the MNIST and FashionMNIST datasets [17] [73]. These are balanced datasets of hand-written characters and clothing items respectively. The term "balanced" means that the dataset has exactly the same number of samples for each class. Obviously these datasets cannot be used for the work described here, but we are informed by their structure and usage, especially for CNNs.

The importance of balanced trainer and validation sets is researched in [10]. This paper investigates the impact of imbalance on classification tasks. The conclusion is that balance is critical for accuracy and that for neural networks, oversampling can be used to correct an imbalance problem. Further, oversampling does not create a problem for neural networks. One of the reasons for their investigation is that researchers often assume that the answer to many problems is to simply acquire more data. Instability in some of our results indicate that a balanced approach is superior. Where possible, the datasets are created without oversampling but in cases where the traffic is harder to come by, oversampling is the method deployed.

2.3 Optimization

When building neural networks, an optimization algorithm is also deployed. Upon computation of the loss calculation for a given iteration, the gradient descent weights are updated during back propagation via some methodology. Optimization techniques vary in how this is accomplished but the point of an optimization algorithm or technique is to arrive at a cost function minimum quickly and accurately.

One of the contributions made with this work is an investigation of optimization techniques. The motivation is quite straight-forward; it is common to see researchers or practitioners stating that there is no reason to use anything other than Adam because it has proven to be so effective. However, we have found this to be a superficial approach to a problem. The best optimizer choice often comes down to the application in question. In the case of packet classification with a variety of neural network types, experiments show that several optimizers perform well.

Typically there are several optimization techniques available in most machine learning packages. The experiment results shown in this work were achieved using the pytorch optim package. Pytorch, available algorithms include Adadelta, Adagrad, Adam (SparseAdam), Adamax, Averaged Stochastic Gradient Descent (ASGD), Limited Broyden Fletcher Goldfarb Shanno (L-BFGS), RMSprop, Rprop and Stochastic Gradient Descent (SGD). Many of the ideas for these techniques have been in practice for decades and some are improvements or variations on a theme. Examples include the addition of adaptive changes or momentum. What follows is a brief discussion of these along with several of the base algorithms.

Stochastic Gradient Decent has certainly been described many times but [21] [24] provide solid background. The pytorch Stochastic Gradient Descent (SGD) has the momentum and Nesterov options. Momentum is a decades long technique for accelerating gradient descent. The idea is that a velocity vector accumulates in "directions of persistent reduction in the objective across iterations" [67]. The momentum updates for the objective function $f(\theta)$:

$$\nu_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t)$$

and

$$\theta_{t+1} = \theta_t + \nu_{t+1}$$

and results showing the efficacy of momentum and Nesterov can be found in the same work [67].

Adagrad [18] or the adaptive gradient method represents a family of algorithms. It is an optimization technique that examines the geometry of the data features and in this way may be able to leverage important yet infrequently occurring details. The authors state that often "*infrequently occurring features are highly informative and discriminative*" and go so far as to state that it is "silly" to have a global learning rate rather than one for each feature. In our case, one concern was that packet traffic offers many features that have little impact but high variance.

Derived from Adagrad, Adadelta [76] is described as an adaptive gradient descent method in that the per-dimension learning rates adjust. This work also reminds us that selecting hyper-parameters such as learning rate can be a bit of an art form. The authors consider the drawback of Adagrad (decay of learning rate, manual global learning rate) as they build in automatic adjustment. The update rule for Adagrad is given as

$$\vec{\Delta} x_t = -\frac{\eta}{\sqrt{\sum_{\tau=1}^{t} g_\tau^2}} g_t$$

where the denominator computes the L^2 norm and η is the global learning rate. Adadelta performs its update over a time window and addresses a parameter mismatch giving:

$$\vec{\Delta} x_t = -\frac{RMS[\vec{\Delta} x]_{t-1}}{RMS[g]_t} g_t$$

The work done with RMSprop is found in a lecture by Geoff Hinton [33]. It is suggested that the learning rates be adjusted at stages during training. In this lecture, several "hints" are given for improving the performance of minibatch gradient descent. These hints include using momentum and RMSprop. Momentum adjusts the calculations in the weight updates to "encourage" the change in the direction of gentle but steady gradients. A Nesterov modification [46] in which large jumps are made followed by adjustment to momentum are made is also included in the pytorch implementation. RMSprop adjusts the learning rate by dividing it by an average of the recent gradients.

Adam (also SparseAdam) [38] is an algorithm that leverages the benefits of two other algorithms (Adagrad and RMSprop). This adaptive moment estimation method updates exponential moving averages gradient and the rate of decay is controlled by other hyper-parameters. Adam "updates exponential moving averages of the gradient and the squared gradient ... the moving averages themselves are estimates of the 1st moment (the mean) and the 2nd raw moment (the uncentered variance) of the gradient." The implemented weight and learning rate update algorithms are:

$$\vec{\alpha}_t = \vec{\alpha} - \frac{\sqrt{1-\beta_2^t}}{1-\beta_2^t}$$

and

$$\vec{\theta}_t = \vec{\theta}_{t-1} - \vec{\alpha}_t \frac{m_t}{\sqrt{v_t} + \epsilon}$$

Adamax is a variant of Adam that addresses instability problems in the update rule for the weights in the presence of large parameter values. This occurs when the L^p norm is used to generalize the L^2 norm.

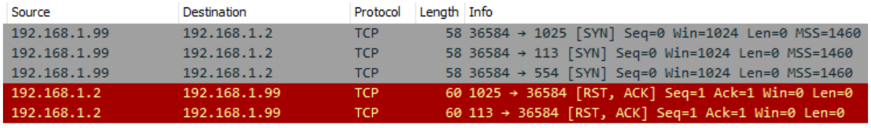
Averaged Stochastic Gradient Descent (ASGD) [51] is one of the more mature ideas in the library and leverages matrix value estimation and averaging to reach optimums. The authors of [51] shared that at the time of that work, this might have been necessary because the optimal algorithm could not be implemented.

An implementation of the Limited Broyden Fletcher Goldfarb Shanno (L-BFGS) method can be found in [7]. The problem of inefficient gradient descent and meta-optimization optimization (hyper-parameter tuning) is addressed by measuring the energy of the descent step and automatically updating the direction and step used.

The Rprop (resilient propagation) approach [56] examines the behavior of the loss or error function. The concern then is not for the gradient itself but the sign of the partial derivative. If the sign changes it is an indication that the step was too large and the update is reduced using an internal factor. If the sign remains the same, the update value is slightly increased. These last three (ASGD, L-BFGS and Rprop) represent some of the older techniques and so have been surpassed by the improvements noted earlier in the section. This statement is supported by the results achieved in this work.

2.4 Security and Port Scans

The need to detect network attacks and anomalous behavior, has been a part of the literature since application weaknesses could be exploited. Historically researchers have long been aware that port scans were important to detect and that detectors easy to evade as there are a variety of challenges associated with detecting scans; both while the scan is in progress and after the fact. An example of the TCP flag behavior during a port scan can be seen in Figure 2.2.



Source	Destination	Protocol	Length	Info
192.168.1.99	192.168.1.2	TCP	58	36584 → 1025 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
192.168.1.99	192.168.1.2	TCP	58	36584 → 113 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
192.168.1.99	192.168.1.2	TCP	58	36584 → 554 [SYN] Seq=0 Win=1024 Len=0 MSS=1460
192.168.1.2	192.168.1.99	TCP	60	1025 → 36584 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
192.168.1.2	192.168.1.99	TCP	60	113 → 36584 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

Figure 2.2: Network Packet

In [64] we see the investigation into backbone scan detection using likelihood variables to determine hypothesis correctness. The variety of hosts, IP diversity and complex routing interferes with successful detection. In addition to this sort of transit location, there are the practical problems associated with even recognizing that a scan is or has happened [65]. Stealth or long term scans require lengthy packet captures, strange flags combinations may get past firewall rules and different types of scans may not be within the scan detector capabilities. Over time a variety of techniques have been deployed though only recently has the increase in processing power allowed a wider variety of machine learning algorithms to be used. Though there is improvement, problems such as false positives still remain [16].

Within machine learning, neural networks with a single hidden layer have proven to be very good universal approximators for a variety of functions [6] and are frequently used to classify images and video components. However works such as [29] and [44] show that neural networks can be effective as packet classifiers. In [14] replicator neural networks and unsupervised learning are used to detect anomalies. In [25] we see a case for the adaptability of neural networks applied to attack detection in mobile devices. [70] uses traditional and convolutional neural networks to distinguish benign from malicious traffic. They achieve a 94.5% true positive rate after training with 1% of available data. Papers such as [61] [49] address port scans and introduce to other variables such as uncertainty and the popular testing tool Snort.

The use of unsupervised learning and sequential hypothesis testing can be seen in [57] although this same work achieves better performance using supervised classification techniques including decision trees and support vector machines [58]. Another approach can be found in [16] in which the authors first collect the desired features and then attempt to separate the observed IP addresses into normal, suspicious and scanner categories by examining behavior in a small duration windows. Another work [2] detects information gathering attempts using neural networks. Attacks are generated using NMAP and their system, once trained, can detect a variety of attacks faster than rule based systems. However, there are some limits to the window observed as it is based on pre-configured number of packets. A similar work [1] builds an Intrusion Detection System though with narrower attack types. In [48] the authors test a number of machine learning algorithms to detect worms in select datasets from CAIDA [13] with an accuracy of approximately 96%.

Importantly, sequential networks that accomplish both the traffic classification and, after separating TCP traffic, address port scans are not part of the literature. The idea of linked neural networks for handwritten character recognition is explored and found to be superior to other methods [11]. Convolutional neural networks and support vector machines are linked to aid in the processing of multi-modal data steams [52]. The ensemble approach can be seen in [74] as researchers achieve high recognition rates with combined convolutional neural networks. For this reason, we have chosen to deploy linked neural networks to break up a very complex task.

2.5 CNNs and Packet Images

While in the past machine learning algorithms have been applied to various networking challenges, only recently has processing power enabled dedicated use of neural networks. Neural networks have been shown to be good approximators for objective functions [6]. Some examples can be found in security research. In [14] replicator neural networks and unsupervised learning are used to detect anomalies and [25] makes the case for neural network adaptability as they are applied to attack detection in mobile devices. [70] uses traditional and convolutional neural networks to distinguish between benign and malicious traffic.

Classification lies at the heart of many network research questions and works such as [29] and [44] show that neural networks can be very effective. Many of these use the artificial neural network or multi-layer perceptron models. An attempt to determine patterns in network traffic by viewing data as images is described in [37]. Using information from the packet headers, images are created that depict address use. From there, anomalies can be detected as the packet streams continue. Over a variety of tests they achieve recognition ranging from 87-95%. Central to these techniques is the convolutional neural network or CNN.

A network traffic classifier that uses convolutional and recurrent neural networks can be found in [41]. In this work the authors use traffic features to process data for the Internet of Things and report accuracies above 96%. However, they do not use raw packet details and utilize unbalanced datasets. The importance of balanced datasets is described in [10]. [20] uses a combination of self-organized maps and convolutional layers applied at various stages in the data processing to detect "divergences in normal patterns". The architecture analyzes images generated from network data and uses netflow data which rather than the actual packet flows. This Netflow sampling is only a small window into actual behavior which means that events or flows can be missed [19].

In [36] the authors attempt to classify encrypted ToR traffic using convolutional neural networks. They process the first 54 bytes (Ethernet, IP and TCP headers) of each packet as input to the model and achieve results ranging from 95-99% depending on the test and classes. A slightly different approach is seen in [40]. In this work the authors work to classify packet types using packet images and the packet payloads across eight applications. They compare a CNN model with ResNet and achieve results exceeding 95%.

As can be seen, researchers are beginning to utilize convolutional neural networks and data presented as images to address networking research questions. We differentiate ourselves by attacking the problems of traffic classification and network visibility by applying convolutional neural networks to full packet images and conversation images created from a collection of packets. An example of a packet converted to an image is shown in Figure 2.3.

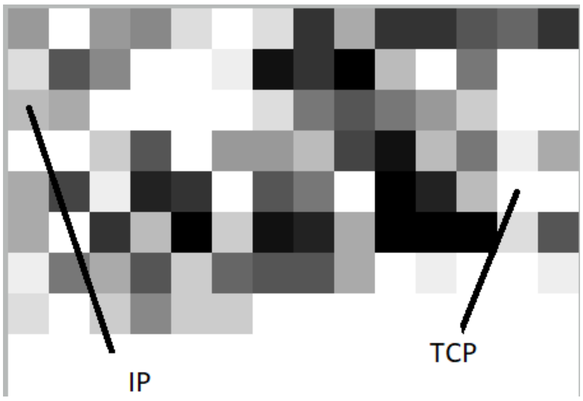


Figure 2.3: Packet as an Image

In both case, raw data is collected from an operating network testbed. A conversation images are analogous to an image with many objects. For these, the

YOLO algorithm [54] processes an image and assigns bonding boxes to objects within. In [55] region based CNNs (R-CNN) are used to achieve near real time object detection. While people staring at an image might not care about all of the objects, a system processing the image might. In the same way, an image made up of packets might be used by a security protocol or quality of service policy. For example, the image might be used as a network based captcha for access control. For problem areas, static images as a aid for visualizing network conditions.

2.6 Ensembles

Machine learning models can be combined in a number of ways. The work described herein uses both MLP and CNN neural networks to classify packets. Once this has been successfully completed, further work uses MLP models sequentially in order to narrow down the focus, and as described in a previous section, address TCP port scans. The subsequent contributions are achieved by combining models into ensembles. A machine learning ensemble is a a collection of models put together in order to achieve more reliable and/or more accurate results. Rather than using the ensemble to find the best hypothesis for our targeted applications, we combine models that are known to perform well in order. This is necessary because high accuracy in the early stages of the described architecture are critical to the success of later stages.

2.6.1 Mean Opinion Score

The primary reason for the ensemble or neural network combinations in this work is to ensure downstream accuracy. The ensemble majority votes on the best classification for a particular sample. Each model is trained on the entirety of the training set. It is not uncommon to achieve 100% recognition rates in the first stage. Subsequently the various next stage group are formed and the next ensemble is trained. For example, after general classes are established, the User Datagram Protocol (UDP) classes are then determined. It is straight-forward to see the relationship between first stage and second stage accuracy.

One of the applications for the UDP stage is the develop a prediction for Mean Opinion Score or MOS. Mean Opinion Score is a qualitative measurement for voice systems and the calculation uses is established in ITU-T G.107 [8].

Because the algorithm uses variables that can easily be derived from a communication packet stream, the MOS can be calculated for Voice over IP [63] [3]. The question we attempt to answer is whether or not a neural network model can be build that can act as a predictor of MOS performance for a given VoIP datastream. Our contribution is that in creating the ensemble for high accuracy packet classification, we can obtain the same high level accuracy for UDP and subsequently leverage this for applications such as MOS prediction.

2.6.2 Adversarial Security

In general, machine learning techniques require access to curated datasets. Otherwise they are prone to misclassification errors. Thus, data that is poisoned, whether by benign accident or through nefarious activity can create a significant challenge to models like those deployed here. Poisoned data is often categorized as adversarial samples. With the increasing use of machine learning models there is a corresponding increase in attacks whose goal is to disrupt model operation. In addition, the adversarial examples need not be markedly different from valid samples - even minor changes can cause multiple models to misclassify the same adversarial samples [22] [68]. Ensembles can be used to reduce model input noise and create diversity that resists misclassification errors [71].

While not investigated directly in this work, this problem space and the successful deployment of ensembles to combat adversarial samples shows another potential application of this research.

Chapter 3

Datasets

3.1 Packet Traffic Data

Access to data is undeniably one of the most difficult challenges for any researcher. An associated challenge is access to data that is both “good” and appropriate for the work being done. In this case, the data in question is raw packet data (actual packets flowing across a network) and "good" means that the data is representative of current protocol use and balanced. Packets can be acquired by either accessing available datasets or by collecting them as part of the project. Both of these methods have weaknesses or barriers.

3.2 Available Datasets

There are many packet datasets available to researchers however, from the perspective of this work they suffer from a number of disqualifying characteristics. The first of these is that many of them are old, dating from 1999-2003. The issue is that over the last 20 years protocol use and operation has changed and these datasets do not necessarily represent current traffic patterns. For example, email is typically accessed via a browser rather than a separate email client. In addition, the volume of traffic utilizing a particular protocol or pathway made have changed. This can be seen in the switch to cloud services which significantly altered the flow paths used. The lack of access to traffic or infrastructure may be the reason that many projects continue to use these datasets. Another reason for their continued use is as a comparison point to previous research. However, at some point, a transition to more contemporary data must be made.

Many of these datasets are highly specific and while this is fine for an objective with the same specificity, it is not suitable for a more general case. For example, a security dataset comprised of port scans cannot be applied to security research covering other vectors. In this case, the researchers would have to compile multiple, and sometimes disparate, datasets. Combining them in this way create meta-data problems with features such as timestamps.

Many of the available datasets are not the packets themselves but feature sets describing the traffic. This is often done to prevent privacy issues but this places the researcher one step away from the actual data. In one case, 248 features about the packet traffic were created and very few of the features represented fields within the packets. In other cases, the traffic is represented by sampling protocols like Netflow. On busy networks, collecting traffic on high capacity links requires that the receiver be able to process 1M packets per second (pps) or more. Even if this is possible, storing the data can also be an issue. For example, a pcap file of 85k packets requires 25MB of storage and a text based file of the same dataset requires 77MB. The Netflow protocol samples packet traffic at intervals and then reports on observed flows. The obvious problem is that this not representative of all flows. Short-lived flows can be missed entirely. The beginning or end of a flow can also be missed.

Some of the available datasets are unlabeled leaving it to the researcher to determine the labeling methodology and classes. Unsupervised means can be used to create clusters around to protocols or addresses but this increases overall error. In addition, there may be no context provided with the dataset so it is difficult to determine the source or event that created the traffic.

It is also the case that the tools and file formats used are incompatible with the current project. As mentioned earlier in this work, neural networks approximate the objective functions using only the raw hexadecimal data. While the file sizes are larger, there is no need for additional software. Datasets not provided in this format require additional conversion which is sometimes incompatible.

Lastly, it is often the case that the datasets are unbalanced meaning that they are heavily weighted towards a subset of the available classes. This can cause a model to overfit or be biased to a particular class. Projects may compensate for this by simply collecting more data. This can be effective but can also result in unstable results, particularly if further imbalanced exists for the other classes.

3.3 Dataset Creation

Given the problems associated with using available data, it was determined that the packet datasets used for this research would be created locally. There were two components to the task; building an infrastructure for the capture and then determining the classes. Privacy concerns precluded raw capture on the campus network. The testbed consists of routers, switches, bare metal computers, hypervisor chassis and a collection of virtual machines. The nodes are observed performing standard tasks and no user specific data is created. Since the variety of packet types is vast, an observation of the most common protocols in use determined the classes. Initially this resulted in twenty-eight classes. The infrastructure is documented in [26] [27]. A portion of the testbed can be seen in Figure 3.1.

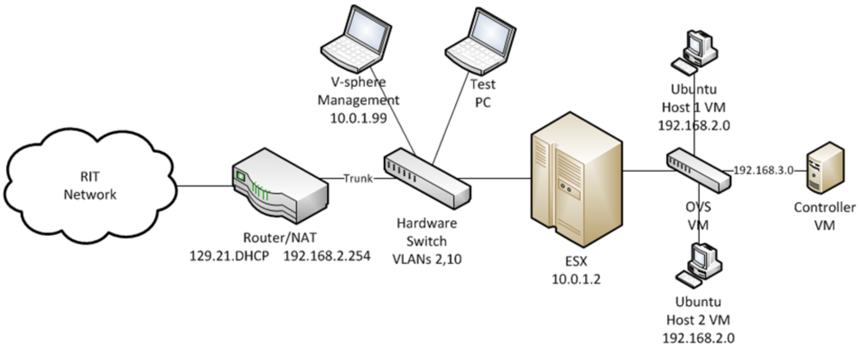


Figure 3.1: Testbed Topology

When capturing traffic with Wireshark, tcpdump, etc., the default format for saving data is often a pcap extension. This requires that the toolchain insert one of these programs. One goal of the work was to allow for native processing and so all datasets were stored in text formats. Each of the aforementioned programs can be directed to save captures in this way however the text files created are structured differently so a Python based parser was written for each one. Within the testbed, tcpdump was most commonly used with the following command: *sudo tcpdump -i eno1 -xx*. In the end, the output of each parser is exactly the same so that the raw, cleaned data can be input directly into the neural networks.

3.4 Balanced vs. Unbalanced

Network traffic is inherently unbalanced in terms of protocol use. The reason for this becomes clear when the applications are considered. Today, much of end user activity occurs within a browser. This automatically directs traffic to a smaller collection of protocols. This is true for web apps, storage, email, and etcetera. Contrasted with the historical methods of accessing data and we see not only a protocol shift but also a weighting towards a collection of these protocols. In the example mentioned earlier in this chapter, port 25 traffic moves to port 80.

At the beginning of this research a survey of protocols used led to the classes or labels shown in Table 3.1. These are broken down by layers.

Table 3.1: "Original Classes"

Layer 2 and 3	Layer 4 UDP	Layer 4 TCP
ARP	UDP QUIC	TCP SSL
802.3 STP	UDP DNS	TCP SIP
802.3 CDP	UDP DHCP	TCP DATA
LLDP	UDP SNMP	IPv6 TCP DATA
ICMP	UDP HSRP	TCP HTTP
IGMP	UDP DATA	
PIM	UDP DHCPv6	
ICMPv6	IPv6 UDP DATA	
IPv6 MULT LIS	UDP SSDP	
802.3 data	UDP SMB	
	UDP DROPBOX	

The unbalanced nature of the protocols can be seen in Table 3.2. These datasets are numbered 29-36 with 29 being used as the large training set and the rest as test sets. This numbering scheme is modified in later chapters after dataset corrections.

Table 3.2: Dataset Classes

-	29	30	31	32	33	34	35	36
STP	4068	1050	14	14	277	395	0	56
IPv4	101863	35612	20230	18853	21657	30869	19504	25500
TCP	80047	24001	19948	18436	19953	29491	14358	25051
HTTP	734	50264	74	371	165	192	82	
HTTPS	14684	4782	1936	5545	2447	7613	3848	5727
UDP	17575	10467	275	386	1604	1269	4262	433
DNS	1854	738	274	374	607	682	899	268
DHCP	106	3	0	0	3	0	5	0
LLMNR	15	40	0	0	8	0	2	0
NBNS	108	20	0	0	0	0	0	0
SSDP	1170	330	0	12	163	190	618	15
IGMP	716	199	0	5	61	71	880	8
ICMP	3165	497	7	6	23	26	0	0
IPv6	1303	263	1	0	129	180	142	0
TCP6	60	0	0	0	0	0	0	0
UDP6	1039	225	1	0	119	174	142	0
ICMP6	204	38	0	0	10	6	0	0
ARP	1632	200	2	0	322	52	1851	447

Clearly, protocol usage is weighted towards a particular set of upper layer protocols. In addition, IPv6 became a problem as the header construction is different from IPv4 in terms of size (40 bytes vs. 20 for IPv4) and field use. For this reason, IPv6 was removed from testing in much of the latter work. It should be noted that classification of IPv6 using neural networks would be handled in the same way as IPv4.

It may be that for some kinds of data, imbalance problems can be addressed with simply adding more data until each class reaches a threshold number of samples. Collecting more does improve results up to a point but the datasets are still imbalanced. Thus, even though classifiers built in the early stages worked very well, there were occasional aberrations in accuracy that could not easily be explained. These aberrations were drops of more than 5%. A closer examination of the datasets led to balancing of the classes within the training and validation datasets. Once this was accomplished, the aberrations of that scale ceased.

The balanced datasets are numbered from 0-5 and include training, validation and test sets. The number of classes were reduced based on the previous experimentation. Several of the class sample numbers are shown in Table 3.3.

Table 3.3: Balanced Dataset Classes

-	0	1	2	3	4	5
1-ARP	5000	500	1851	447	200	0
2-ICMP	5000	500	0	0	25	0
3-IGMP	5000	500	880	8	199	2356
4-Loop	5000	500	0	0	0	2907
5-STP	5000	500	0	56	1050	14534
6-CDP	5000	500	0	2	32	484
7-DNS	5000	500	899	268	738	868
8-DHCP	5000	500	5	0	3	67
9-SSDP	5000	500	618	15	330	3111
10-NBNS	5000	500	0	0	20	27
11-80	5000	500	659	618	11931	0
12-8080	5000	500	567	0	1154	0
13-443	5000	500	13131	24433	12463	0

Based on performance testing, the number of samples for each class was set at 5000. Lower sample quantities did not resolve the instability problem and larger sample quantities did not provide increased accuracy for the increased training time. The validation set is 10% of the training set size and so each class has 500 samples. The test datasets have random sizes for each class as they are raw network captures.

3.5 Why packet datasets are different

In the search for balanced datasets there are a variety of techniques that can be employed. In the example of image classification, a single image can be manipulated in many ways in order to increase the number of samples. For Images can be reversed, rotated, inverted, colorized or skewed. If the concern is only for packet classification, similar techniques can be applied to the traffic datasets. That is, packets can be copied or manipulated so that they are still valid in order to increase the number of samples of a particular traffic class. Many of the other classes (such as TCP port 80) need not be manipulated in this way because there are plenty of samples.

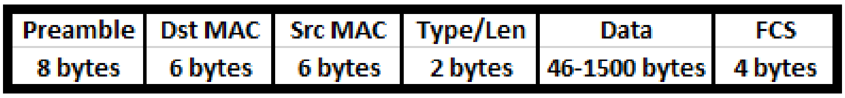
However, there is a limit to what can be done with packets manipulated in this way. This is because packets do not operate in isolation from each other. It is often the relationship between packets that must be used in order to understand what is happening in a particular flow. As models attempt to address visibility or specific problems such as security or application performance, this timestamp is used directly. In these cases, the timestamp is critical and packets taken out of sequence or manipulated end up affecting the model operation, labeling or downstream processes. This problem worsens when packets are sorted by class due to dataset or capture assembly as was the case for the port scan detection and Mean Opinion Score experimentation described later in this work. The impact is to require the creation of additional custom datasets. In the Mean Opinion Score prediction model, Real Time Transport Protocol (RTP) streams were required. The model had to find sequential packets of the same stream rather than simply classifying RTP packets.

3.6 Significant Features

It is also common to apply dimensionality reduction or compression techniques to help speed the machine learning training process. Principal Component Analysis (PCA) is widely used to determine the features containing the greatest variance which can then be used to accurately classify the object of interest [72]. However, depending on the type of data, PCA may not achieve the desired results due to the loss of potentially important features. More details regarding the efficacy of employing PCA to the problem of network traffic classification are discussed in Chapter 5.

3.7 Construction

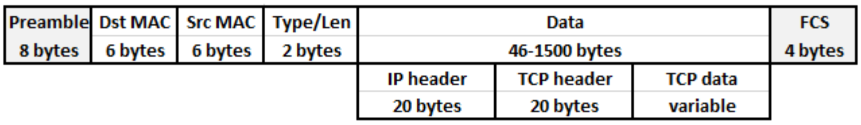
Dataset size is a function of the number of packets and the features used by a particular model. The sample features are the packet bytes from raw capture. An Ethernet frame has a maximum size after capture of 1514 bytes (1526 before) which are organized into several fields. Of course, the machine learning model does not process the fields. An Ethernet frame is shown in Figure 3.2.



Preamble	Dst MAC	Src MAC	Type/Len	Data	FCS
8 bytes	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

Figure 3.2: Ethernet Frame

It should be noted that there are actually two types of Ethernet frame; 802.3 and Ethernet Type II (shown). The two run at the same time and the neural network model handles either type. The data field encapsulates the information and upper layer protocols. When an Ethernet frame is transmitted, the preamble field and the frame check sequence are not seen by the capture software since they are removed after processing by the network interface card (NIC). This means that for the features, the count begins with the destination MAC address and ends with the data field. An example of a frame encapsulating IPv4/TCP packet is shown in Figure 3.3. The grayed out areas are the fields not present after processing by the NIC.



Preamble	Dst MAC	Src MAC	Type/Len	Data	FCS
8 bytes	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes

IP header	TCP header	TCP data
20 bytes	20 bytes	variable

Figure 3.3: TCP Encapsulation

All of the packet analyzers used here save packet data in the form of hexadecimal characters with two characters being equal to one byte. The features chosen refer to the number of hexadecimal characters desired. In the example shown, choosing 108 features (54 bytes) would encompass the three headers only. Over the course of testing with multi-layer perceptron models features sizes of 125, 150, 175 and 200 were tested. In the latter portion of this work with convolutional neural networks and a slightly increased number of classes, 196 features were used resulting in a training set matrix size of 196x85000. The 85000 refers to the number of training samples/packets.

3.8 Minimum and Maximum size

The data field of an Ethernet frame is a variable size ranging from 46-1500 bytes. Encapsulated Internet Protocol (IP) packets are also variable in size. This means that upon feature selection, a short frame may not have enough data to provide the necessary bytes and a long frame will have to be truncated. In cases like this, the short frame is padded with zeros. Frame truncation means that user or application data is sacrificed. This typically does not create problems for packet classification. This is because the headers containing identification fields remain. For example the Ethernet Type field describes what is encapsulated in the data field. Within the IP header the protocol ID field indicates the next level of encapsulation.

Decision trees and table lookups use protocol header field information to help with identification. While the models do not "read" the fields, they do recognize the patterns. These and other patterns that exist within the frame/packet structure can be recognized. For example, an ARP request packet has a 6 bytes pattern of zeros in the target hardware field. It is these patterns that are leveraged by a function approximator such as a neural network. For this reason, the models described throughout this research do not truncate before the header fields appear.

Chapter 4

Neural Networks

4.1 Neural Networks

The Multi-layer Perceptron (MLP) and Convolutional Neural Network (CNN) models deployed in this research are described in this chapter. In addition to single neural networks ensembles and sequential architectures of neural networks have also been created. Currently the ensembles are linked sequentially although they could be run on different graphics processors simultaneously. The weight matrices can also be saved however, the training sets have consistently changed making this impractical for many of the investigations.

4.2 Multi-layer Perceptron Neural Network

Initially and as part of the ensembles, we have used Multi-layer Perceptron Neural Networks (MLP NNs) because they have been shown to be good function approximators. MLP neural networks are generally categorized by the number of layers they contain; an input layer, one or more hidden layers and an output layer. Input layer size is dependent on the number of features used in testing. After parsing the capture files, input feature selection is a matter of selecting the desired number of hexadecimal characters (input bytes * 2) from each packet. The number of input features has been extensively investigated over the course of this work and details are provided in subsequent chapters. The hidden layers vary in number and size (hidden nodes per layer) as the optimal structure for any particular objective function is usually discovered via experimentation [9]. Generally speaking, as problems become increasingly complex or the number of classes increases, the structure of the network also becomes more complex. The output layer is dependent on the number of classes or labels. Some of the architectures used here are shown in Figure 4.1.

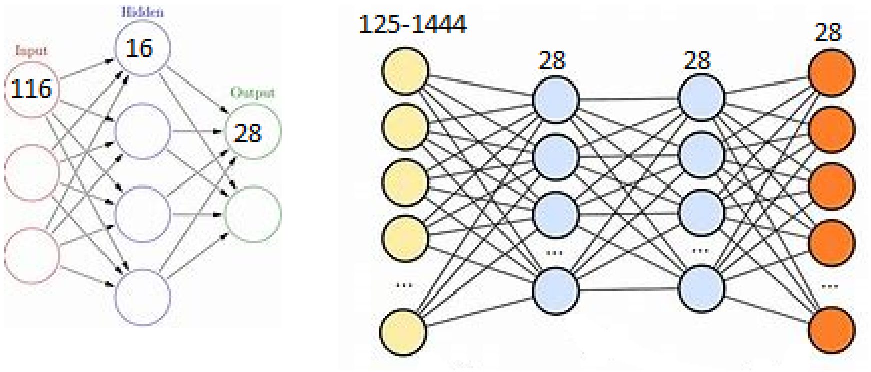


Figure 4.1: Multi-layer Perceptron Models

Each layer is comprised of nodes. Moving through the layers, nodes represent linear combinations of the previous layer and these are passed through an activation function which is used to determine the output of a particular node. Activation functions include the sigmoid, tanh and ReLU (rectified linear unit) functions. The early models used the tanh function similar to the work done by [44]. The output value of each node is between -1 and 1. Later experiments adopted the ReLU function which forces the output between 0 and 1. Graphs of the activation functions can be seen in Figure 4.2

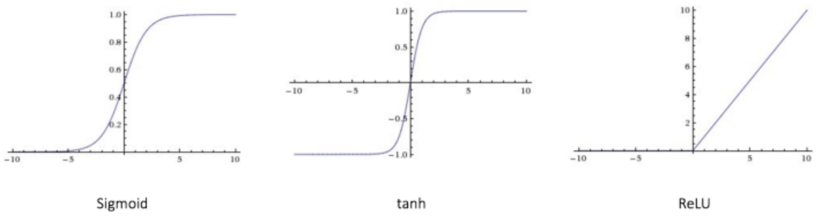


Figure 4.2: Activation Functions

All of the calculations are on matrices where matrix X=[features, packets] and W is the weight matrix with the same dimensions: $X = [x_0, x_1, ..., x_n]$,$W = [w_0, w_1, ..., w_n]$. Starting weights in W are random. After multiplying the matrices, the values are passed through the activation function (g). The result is a matrix commonly denoted Z: $Z = [z_0, z_1, ..., z_n]$. Thus the calculation for each node becomes:

$$h_0 = g(x_0 w_0 + ... + x_n w_n) = \sum_{i=0}^{n} x_i w_i$$

Each layer performs these calculations. At the last layer a probability for the correct label is established, often via a Softmax function. Softmax takes the resultant values (Z) from the previous layer and calculates the probability as follows:

$$Softmax = exp(Z_i) / \sum_{i=0}^{n} exp(Z_i)$$

The probabilities are compared to the ground truth labels to produce an error or loss.

At this point, forward propagation through the neural network is complete and now the error is backward propagated through the network. Back propagation updates the weights used in the forward calculations. This process is called gradient descent (GD) which uses a learning rate and partial derivatives to take a step in the direction of the loss function minimum. There are variations on this algorithm that range from using all of the training samples in each step (batch GD), a single training sample in a step (stochastic GD) or a subset of the training set in each step (mini-batch GD). The trade-off between these is in the training time vs. smooth performance and accuracy. Generally batch GD takes much longer while stochastic GD has a more erratic path to the minimum. Figure 4.3 depicts batch gradient descent over 1000 iterations. From this image we can see that batch gradient descent is monotonically decreasing.

Figure 4.4 shows the results of stochastic gradient descent and indicates that stochastic is much less predictable meaning that the error can vary significantly from iteration to iteration. For clarity the inset shows behavior for stochastic gradient descent over the first 100 iterations.

A similar test was run with mini-batch GD. This is shown in Figure 4.5. With the training time and accuracy improvements, a batch size is 128 was found to be very effective. In addition to using mini-batch, an optimizer can be

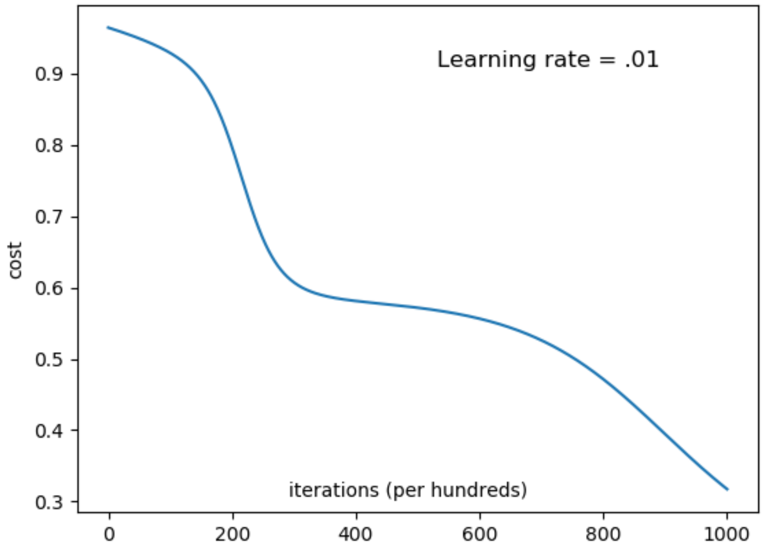


Figure 4.3: Batch Gradient Descent

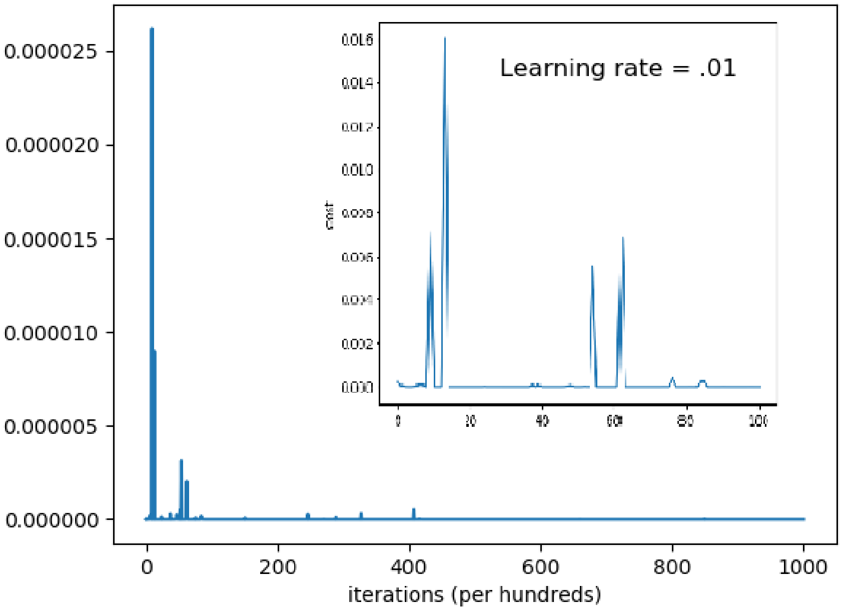


Figure 4.4: Stochastic Gradient Descent

deployed which improves the performance (training time, accuracy, stability) of the neural network. Optimizers often include techniques to manipulate the weights or learning rates. The example shown in figure 4.5 were achieved after adding a weight regularization term. Previous explorations reveal that the use of mini-batch speeds training time (cutting the stochastic time by just under 75%) and the addition of the regularization term increases accuracy by 1%.

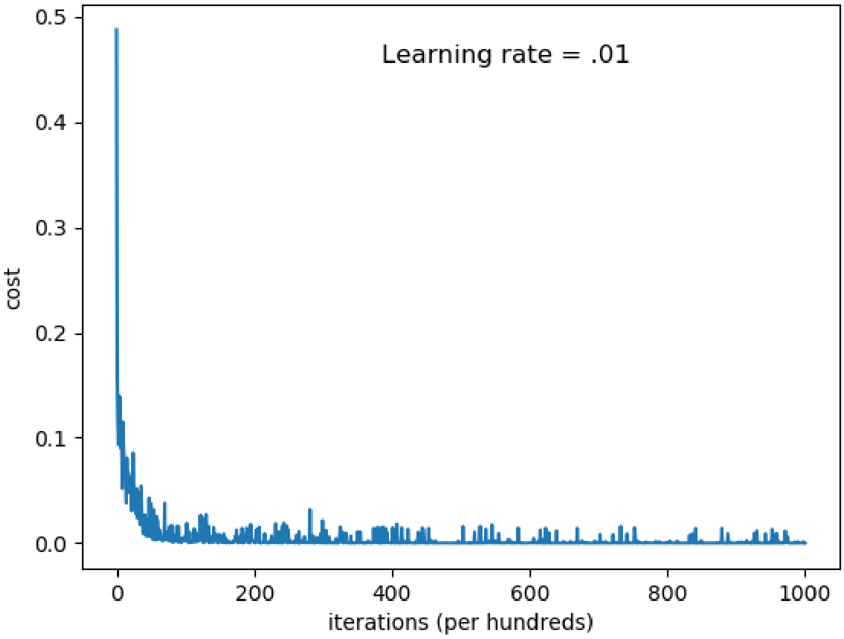


Figure 4.5: Mini-batch with Regularization

Backpropagation works back through the neural network to update the weight vectors (based on the error calculation) via mini-batch gradient descent. The forward propagation is then run again changing all of the linear combinations and the error or loss recalculated. This recalculates the probabilities associated with an output label. The escape condition is either completing the desired number of iterations (ex. 5,000) or reaching a particular loss value (ex. 1E-8). As the process continues, one of the output classes develops a high probability of being the correct label and in this way, the neural network learns the how to correctly identify the incoming packets.

In addition to structure, the success of any network is often tied to the hyper-parameters (learning rate, batch size) and the quality of the training data. Later chapters will detail the ranges investigated for each of these.

The initial neural network used for packet traffic classification varied quite a bit from final models. The initial model had 116 input features, a single fully connected hidden layer with 16 nodes, an output layer size of 28 and used batch processing. Subsequent architectures added hidden layers, varied the number of input features and used mini-batch. For example, extensive testing with a variety of learning rates and hidden layer sizes of 20 and 36 in order to determine the impact of such changes was completed. The initial and final multi-layer perceptron models for early classification work are reflected Figure 4.1. After extensive testing, the final size of the hidden layers was set at 28 nodes, with input features of 125/150, 128 batch size and output classes dependent upon the experiment (general classification, TCP, UDP) currently being run.

4.3 Convolutional Neural Network

Convolutional neural networks differ in that the input object is typically an image rather than a vector. One of the unique contributions of this work is that packets are re-imagined as images and a packet stream can be thought of as a video. One aspect of a convolutional neural network is that it reduces the number of features between layers. For example, a pair of fully connected (FC) layers is a linear combination of all nodes in one layer to all of the nodes in the next layer. The convolutions and max-pooling used in CNNs significantly reduce this number which can make the CNN faster than the MLP. During the CNN forward pass, the packet matrix is run through a series of convolutional and fully-connected layers. CNNs accomplish this through the use of filters which pass over the image and apply a transform for each step of the pass. The transform result is also a linear combination.

For example, the first convolutional stage of our network uses a 3x3 filter and a step of one. The filter is made up of a grid of weights. The 3x3 filter (f) starts in the upper left hand corner of the packet image covers a 3x3 portion of the 14x14 image. The filter then steps to the right until it reaches the far edge. The filter eventually returns to the left moving down as needed to eventually pass over the entire original image making calculations as it goes. The size of the step is referred to as the stride (s). The result is a new 12x12 image. We do not use any padding (p) for the images. The resultant image size can be determined by the formula:

$$((n - f + p)/s + 1)$$

For these packet images: ((14-3+0)/1 + 1) = 12. Following the convolutional layer (conv1), a max-pooling layer further reduces the number of features. The max-pooling layer performs the same passes, though the filter is smaller (2x2) and the stride is doubled. The value obtained from each step is the maximum (argmax) for the portion of the image covered by the filter. After the first convolutional and max-pooling layers have made their passes, the resultant image is 6x6: ((12-2+0)/2)+1=6.

This process is repeated with a series of additional filters. Each filter does the same thing though their weights are initialized separately. Filter calculations are combined later in the model. As an example, the first layers of our model use six filters and the subsequent layer uses 16. Another smaller example of a CNN with max-pooling and a series of filters is shown in Figure 4.6. For space, the image starts as a 10x10, is reduced to 8x8 ((10-3)/1+1=8) and then after max-pooling with a 2x2 filter the image is 4x4. There are also six different filters applied to each image.

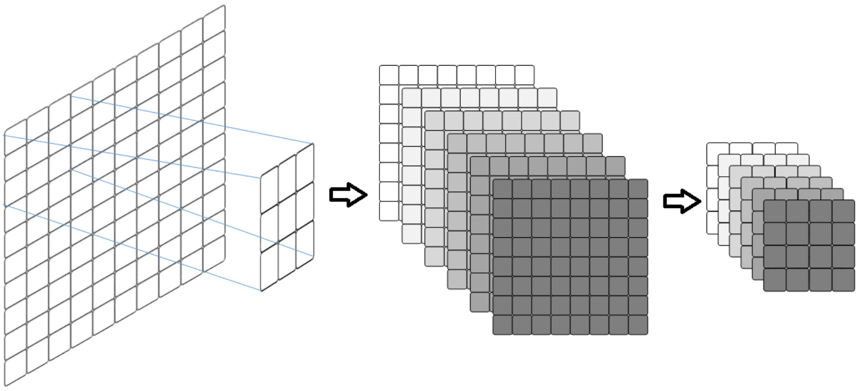


Figure 4.6: Convolution with max-pooling

As the actual 14x14 packet images are small, the first stage is only followed by a one other stage (conv2, max-pooling) which further reduces the images to 2x2. Because of the additional filters, there are a number of these results for each packet. These results are then combined in a series of fully-connected (FC) layers. Fully connected layers rearrange the data once again to a column vector. The column vector size is the product of the image resolution and the number of filters. For this configuration: 2*2*16 filters=64. A series of fully connected layers reduces this value to the number of classes.

An example of this part of the network can be seen in Figure 4.7. Continuing the smaller example, the 4x4x6 images would be combined into a 96x1 column vector which would be further reduced to 14x1 in our case where 14 is the number of classes.

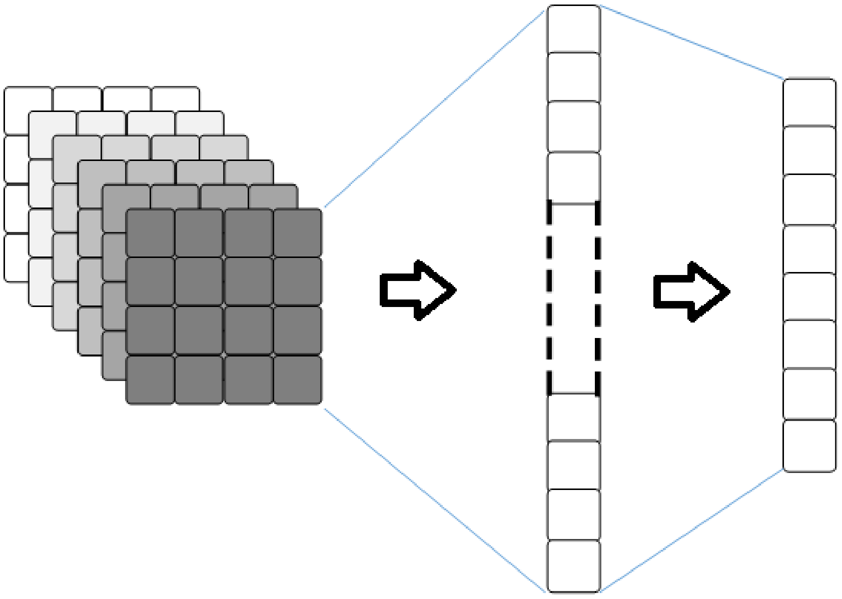


Figure 4.7: Fully connected layers

At the end of the forward pass, a Softmax function then calculates a class probability. Once the error is calculated and back-propagated, the model is run again until either the desired number of iterations has passed or the error rate has declined below an acceptable threshold. The full model can be described as conv1 (3x3x6), max-pooling (2x2, stride=2), conv2 (3x3x16), max-pooling (2x2, stride=2), FC1(64:32), FC2(32:16), FC3(16:14). Training typically runs 1000 to 2000 iterations. While the training is off-line, the architecture itself can then be used on stored datasets or during real-time captures.

Details rationale regarding the choices made in the construction of the CNN will be described in greater detail in subsequent chapters.

4.4 Ensembles

In this work, ensembles have been used for three different applications. The first was to address the challenge of port scans. The networks are connected together with the first being a general classifier and the second being used as a TCP classifier. The architecture depicting the NN inter-connection is shown in Figure 4.8. An input feature vector of the first 150 hexadecimal characters from each packet performs well for general classification. This value is used for both stages. Currently the general classifier has 13 general classes or labels. The TCP packets are separated and used as input into the TCP stage so that the TCP flag combinations can be determined. This operation is more fully described in Chapter 7.

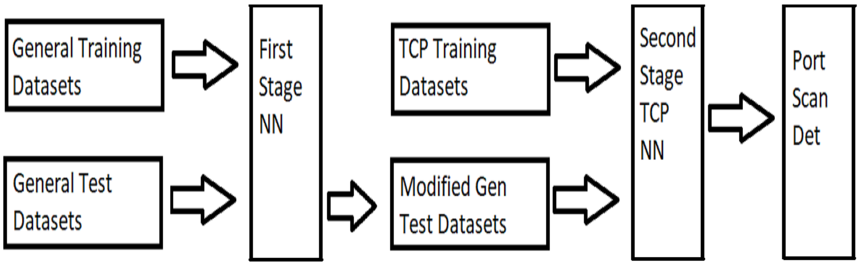


Figure 4.8: Model Architecture

Ensembles are more fully explored in 9 as a collection of models perform general classification followed by additional TCP and UDP classifiers. Subsequently the UDP packets are further separated for VoIP processing. For this application, the RTP packets are input into another neural network that predicts Mean Opinion Score for the RTP streams. A portion of this architecture is shown in 4.9.

The third potential application for the neural network ensemble is to help combat adversaries wishing to either poison the data or fool the network network

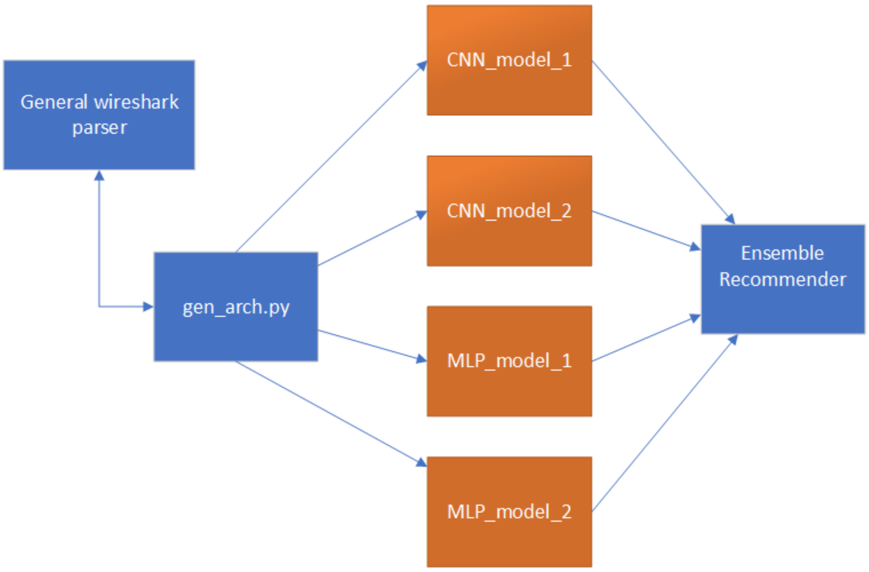


Figure 4.9: Larger Ensemble

structure [71]. In these attempts, the training or test data can be manipulated order to impact the machine learning algorithm output. The neural network ensemble combats this by applying multiple models to the data in order to more accurately determine the correct class.

Chapter 5

Multi-layer Perceptron Classifier

5.1 Preprocessing

Like most machine learning systems, incoming packet data must be preprocessed. Preprocessing is the "cleaning" of the data to remove unwanted artifacts. Programs such as Wireshark [47] and tcpdump [42] add a good deal of extraneous information to the saved text files. If the capture files are not available, additional processing or analysis is required. The flow datasets used in [45] are derived from pcap files but do not provide the actual captures. This dataset also has several meta-features such as arrival times. All of the features are organized into .arff files that are input into the Weka toolset [34]. Processing these files is straight-forward but the challenge is in determining which fields are to be used. Given that [4] [44] focused on TCP, many of the features are constant. In addition, fields will contain zeros if the feature was not observed. While this may occasionally help in classification, excessive zeros may make it more difficult.

Preprocessing is followed by feature selection and normalization. Both of these are discussed later in this section. Another step in the processing is to extract the ground truth label for the traffic. This label is used in the error calculation as the neural network attempts to properly classify the training and test sets. When initially developing a comparison to [4] [44], all of the 248 features are used and then this number is reduced this number until similar results are achieved - ending with a small (22) set features. A majority of the eliminated features had been set to zero in the datasets.

Moving forward, this study departs from meta-data classification attempts because even though raw packet captures can be expensive in terms of space and processing, they are far more accurate. An example of a misleading meta-data feature can be found in packet size. Packet size is an common consideration when attempting to recognize packet type. For example, ARP messages are typically much smaller than those carrying user data. Though there is some variation within the capture datasets, average packet size is 600-700 bytes. While this can provide some insight into traffic categories during a particular time frame, it does not reveal much with regard to individual packets.

Examining packet sizes in one sample of 100K packets, 50% were in the range of 40-159 bytes and another 34% were greater than 1280. Even with this number of small packets, it is not reasonable to assume that they are remotely similar. Other small packets are padded to change their size to the minimum 46 bytes required for Ethernet transmission. This analysis underscores the reasoning for retaining actual packet bytes. Thus, this approach requires a greater number of meta-data features. However, each one has the same weakness. We contend that meta-features are not sufficient for packet classification, especially if the packet classifier is to be used in downstream applications such as security.

As a result, the large number values or fields captured in the meta-data are simply not needed. The storage recovered compensates for the size of the capture files. The timeliness of the packet captures can also provide greater insight into actual transactions on the network. In our case, the data center testbed traffic was captured using command line versions of the Wireshark [47] and tcpdump [42] tools. This allows us to stay in the shell environment, gives access to real time data and eliminates the need for a GUI based program.

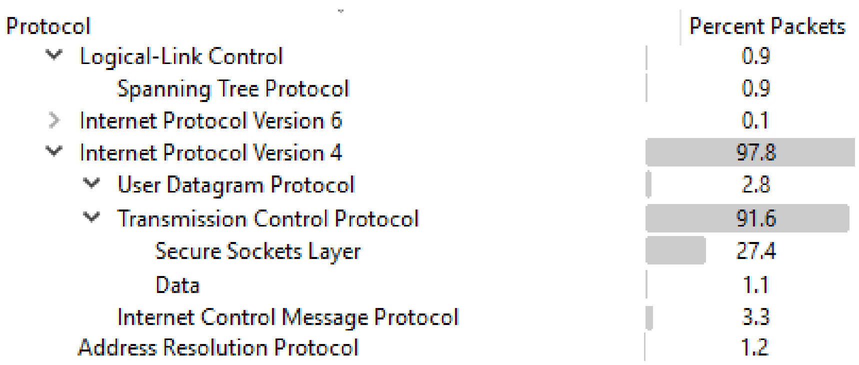
Even with the actual packets, choosing the correct features can have a vast impact on both the accuracy and time needed for operation. Unlike the Weka data files, the choice is made to save all of the features (bytes) for packet classification, taking the raw bytes from the packet capture and then experimenting with how many are necessary to properly classify individual packets. Our MLP comparison was over input feature sizes of 125, 150, 175, 200, 1444 hex characters. In this way, the effectiveness of adding more header fields and packet content, eventually maxing out at nearly a full packet can be determined.

The minimum Ethernet frame size of 64 bytes corresponds to 128 hex characters. As correctly pointed out in [39] the trade-off is between a greater number of features for accuracy against the need for timeliness and reduced processing. It is important to note that when input feature size is selected, any packet smaller than that will require some form of padding. In this case, small packets are padded with additional zeros consistent with current RFCs.

For flow classification, the focus is on tuples normally used to identify flows [12]. These include source and destination addressing, ports and protocol IDs. In some of the work cited here, authors have treated complete TCP flows only. That is, flows that include the identifiable TCP setup and teardown handshakes. However mid-flow packets may be just as important. In addition, it may be that a flow is setup via a completely different conversation. For example, real time protocol streams (RTP) are established via the Session Initiation Protocol (SIP) conversation. Thus it is important to be able to detect partial flows as they can have a significant impact on performance or current conditions. Lastly, because packet captures show that so much of contemporary network traffic is not TCP based 5.1, we also provide UDP, ICMP and IPv6 processing.

A determination of ground truth must also be established for each sample. With image recognition humans are often employed to complete the very manual task of classification. This is also the case with [45]. However, communication network traffic is deterministic with the fields and headers describing the identification of each packet. Thus, during the preprocessing stage, the architecture able to parse the file and determine the ground truth in this way. This is verified with the neural network recognizing 99.9% of the training packets and a manual review of the packets via the original capture.

This classification effort is focused on experimenting with a representative set of contemporary classes. To this end a variety of captures were analyzed to determine the most common protocols in use. While captures vary, the classes chosen cover the typical collection of layer 2, 3 and 4 traffic types. Figure 5.1 depicts a sample from a 400k packet protocol distribution.



Protocol	Percent Packets
Logical-Link Control	0.9
Spanning Tree Protocol	0.9
Internet Protocol Version 6	0.1
Internet Protocol Version 4	97.8
User Datagram Protocol	2.8
Transmission Control Protocol	91.6
Secure Sockets Layer	27.4
Data	1.1
Internet Control Message Protocol	3.3
Address Resolution Protocol	1.2

Figure 5.1: Packet Distribution

Currently testing uses twenty eight classes though it is clear that the number of classes will grow depending on applications and services. Table 5.1 details these classes.

Table 5.1: Current Classes

Layer 2 and 3	Layer 3 UDP	Layer 3 TCP
ARP	UDP QUIC	TCP SSL
802.3 STP	UDP DNS	TCP SIP
802.3 CDP	UDP DHCP	TCP DATA
LLDP	UDP SNMP	IPv6 TCP DATA
ICMP	UDP HSRP	TCP HTTP
IGMP	UDP DATA	
PIM	UDP DHCPv6	
ICMPv6	IPv6 UDP DATA	
IPv6 MULT LIS	UDP SSDP	
802.3 data	UDP SMB	
	UDP DROPBOX	

As discussed earlier, these classes evolved as the system matured and so these represent an early set of classes.

A fair question at this point is whether or not the neural network is needed if Wireshark already provides the information. In addition, if the python based parser can determine the information, why bother with the more complicated neural network? The answer is twofold: first, neural networks are very fast once trained and the weights established. A graph showing this comparison can be seen in 5.2. The Wireshark timing measures simple file opening time for the text files in use. It should be noted that additional time is taken for any analysis (ex. protocol distribution) inside Wireshark. The CNN processes datasets in slightly less time than Wireshark but the MLP is faster by more than a factor of ten. Both of these performance improvements were achieved without the need for a separate protocol analyzer. It should be noted that the neural network datasets are cleaned versions of the original capture files and so some time was expended in the preprocessing. However, this can be done offline and does not impact the speed improvement shown here.

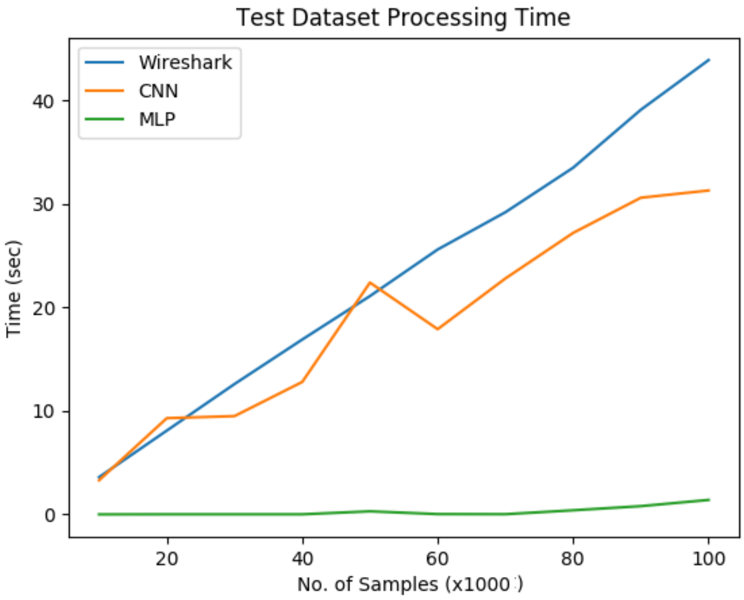


Figure 5.2: Dataset Processing Time Comparison

Secondly, the end goal is not necessarily to be able to identify packets (although this may be desirable) but to be able to identify flows, network conditions and develop applications for machine learning. Neural networks can provide a much richer form of classification and learning than simply parsing through a text file. Our goal is to eventually be able to use communication traffic as *one* of several inputs into a system and this represents a first step toward that end. As described in [78] and [43], these are aspects of a broader picture representing a new paradigm for communication networks. As can be seen in the later chapters, a number of applications have been developed that all begin with the packet classification problem.

5.2 Dimensionality Reduction

As training a neural network can be time consuming, incoming data is analyzed in order to determine the most effective features. This is because patterns in the data can be found without using all possible features. Principal Component Analysis (PCA) is a dimensionality reduction or compression tool that can speed the training process by selecting the features with the most impact for classification. It can also be used to select the best two or three principal components so that a dataset can be visualized. Recently some practitioners have suggested that achieving the performance goals of a particular model may not require the extra step taken for PCA. Here Single Vector Decomposition (SVD) and the subsequent Truncated SVD are used to reduce features used in processing the datasets.

Most of the critical fields for traffic classification (source and destination IP address, source and destination ports, protocol ID, type) occur within the first 66 bytes of a packet. This includes the entirety of the Ethernet, IP and TCP headers. Some of the Ethernet fields (preamble - 8 bytes, frame check sequence - 4 bytes) are dropped after processing by the network interface card. This changes the TCP packet size to 54 bytes and UDP to 42 bytes. For TCP based traffic this would mean 108 hexadecimal characters. However, there may be occasions where other fields or packet content (ex. higher level application headers) may assist the classification effort. For UDP, 84 hexadecimal characters would be used.

The concern when using PCA for packet traffic is two-fold; information loss and behavior between datasets. While PCA does not always result in a loss of important information, it can. This concern arose when PCA reduced the number of dimensions well below what was thought was critical for accurate classification. For example, with 200 input features, PCA reduced this down to 92 (46 bytes) which was fewer than the header fields. In addition, larger feature size selection were a concern because the excessive amount of padding on smaller packets might increase the variation and over-emphasizing field importance. In the different datasets, packet content can vary wildly and so change the emphasis of various features. Lastly, there are a collection of fields that have a great deal of variability but have little effect on the classification. The header checksum and identification fields in the IP header are perfect examples.

The PCA performance in the early experimentation was perhaps the most surprising result in that it performed so poorly. Based on the early feature reduction percentages seen, several values were chosen that would provide the necessary number of features while allowing the compression from PCA. The neural network was allowed to run using the highest performing parameters including batch sizes of 64 and 128, a learning rate of .01 and a hidden layer of 28 nodes. Like the other tests, the network completed 5000 iterations. The tests varied the input features over the values of 256, 384, 512, 1024 and 2048. For space, only the top four recognition rates are shown. The k value is the number of features left after the transform is complete. Like the earlier tests, the reduction can exceed 50%. The recognition rates for the test set are extremely low. With a larger number of input features, there is a near doubling of this value but this comes with an alarming increase in the training time, negating the reason for using PCA. When PCA does not accomplish the desired improvement in training time it may be that the training data is not well formed for the purpose. In this case, it is likely that there are fields included that may not be necessary and that actually skew the results. In addition, smaller packets are often padded which may create the illusion of variance where there is none, also skewing the results.

Table 5.2: PCA results

Features	Batch Size	k	Recog Rate	Time (min)
256	64	126	.253	269
256	128	126	.254	259
2048	64	1289	.427	530
2048	128	1289	.379	640

Over the course of the work, datasets were improved and the classes modified to suite newer models. Dimensionality reduction was revisited using TruncatedSVD from scikit-learn.org. In Figure 5.3 some of the results both with and without PCA are shown. These graphs depicts the results after applying TruncatedSVD to the datasets at 99, 95, 90 and 85% of the 108 features (54 bytes) available. The results without dimension reduction are shown for comparison. The left-side graph contains the raw results. On the right, one of the worst performing datasets was removed. In addition, since IP and MAC addressing are some of the fields that can vary without providing much detail for classification, these fields were zeroed. As can be seen in both, dimensionality reduction did not help with model performance. In addition, training time was only improved by an average of 17 seconds with the addressing fields and 3 seconds without. These results lead to the conclusion that dimensionality reduction techniques are not effective with packet datasets. The cause for this is likely in the variability of fields that can sway the results of an algorithm calculating significance based in part on this variation.

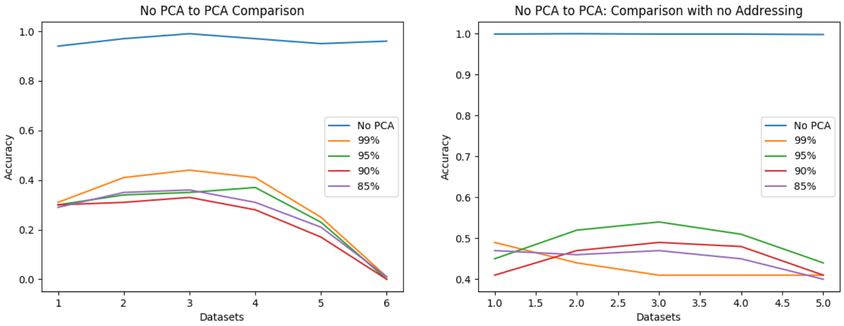


Figure 5.3: No PCA to PCA comparisons

5.3 Multi-layer Perceptron Models for Traffic and Flow Classification

The structure and operation of the MLP neural networks used in this and the following chapters are explored in Chapter 4. As a reminder regarding this early stage of the work, the tanh activation function is used while and later work used ReLU. The neural network input layer receives selected features and the output layer size is the number of possible classes.

The original 28 classes (see Table 5.1) include layer 2 specific protocols such as

Link Layer Discovery Protocol (LLDP) and the Spanning Tree (STP) and layer 3 protocols such as IPv4 and IPv6. Classes have also been allocated for protocols such as the Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). Common applications have also been identified.

The initial neural network used for network traffic classification varied quite a bit from our final model as it had 116 input features, a single fully connected hidden layer with 16 nodes, an output layer size of 28 and used batch processing. Subsequent architectures added hidden layers, varied the number of input features and used mini-batch. Our initial and final multi-layer perceptron models are shown in Figure 5.4.

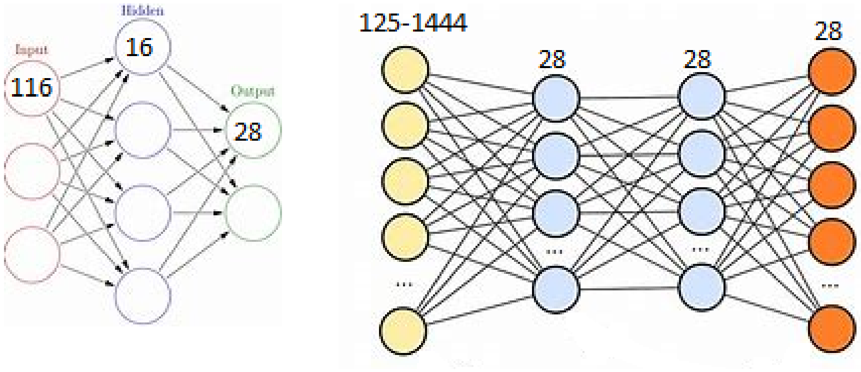


Figure 5.4: Multi-layer Perceptron Models

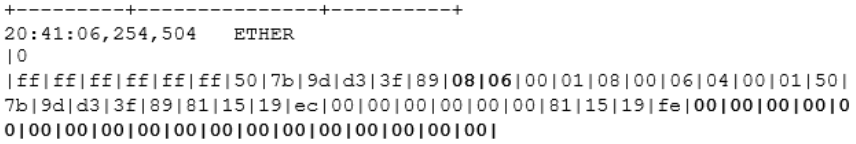
Minibatch sizes were 64, 128, 256 and 512. These results are found in Table 5.3. With these layers, there are two weight calculations in a forward propagation phase and both are updated during backpropagation. After extensive testing, the final structure added nodes to the hidden layers to a total of 28, with the same values used for input features, batch sizes and output classes. However, extensive testing with a variety of learning rates and hidden layer sizes of 20 and 36 was performed in order to determine the impact of such changes. The following sections present our method for preprocessing data and obtaining features, class determination and results based on the structure of the neural network.

5.3.1 Datasets

Much of the previous work cited used datasets that might be considered out of date. Datasets must also have enough data to ensure that the neural network can be trained without overfitting. Another major concern was privacy. Recall that to address these general problems, a testbed was constructed that contained desktop computers, virtual machines and containers. The testbed and some earlier work are documented in [27]. The challenges associated with datasets are fully explored in Chapter 3.

5.4 Operation

The procedures described in this section apply to both MLP and CNN models. To begin, a single sample entry is examined. Shown in Figure 5.5, it is from a text file written with a program like Wireshark.

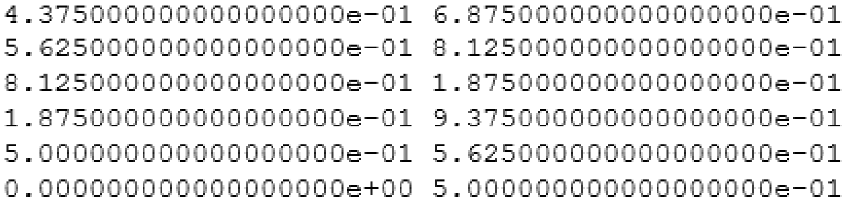


```
+---------+---------------+----------+
20:41:06,254,504   ETHER
|0
|ff|ff|ff|ff|ff|ff|50|7b|9d|d3|3f|89|08|06|00|01|08|00|06|04|00|01|50|
7b|9d|d3|3f|89|81|15|19|ec|00|00|00|00|00|00|81|15|19|fe|00|00|00|00|0
0|00|00|00|00|00|00|00|00|00|00|00|00|00|
```

Figure 5.5: Wireshark Packet

This example depicts an ARP message which can be determined by some of the labeling fields (0x0806) noted in the previous sections. The second set of bold face indicates the padding used here to create the minimum frame size necessary. This same type of padding is employed when extending the standardized packet size for the neural network. A processed version of this entry is shown in Figure 5.6. Due to length constraints, only a portion is shown.

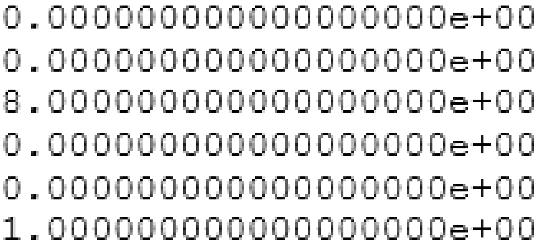
For this neural network, the number of X input features can be set to 125, 150, 175, 200 and 1444. These values, while not exhaustive, were chosen to test intervals containing the major label fields. The effect of these values is to vary the size of each packet and therefore the number of features for input. For example, at 125 features, the layer 2 header (14 bytes), layer 3 header (20



```
4.375000000000000000e-01 6.875000000000000000e-01
5.625000000000000000e-01 8.125000000000000000e-01
8.125000000000000000e-01 1.875000000000000000e-01
1.875000000000000000e-01 9.375000000000000000e-01
5.000000000000000000e-01 5.625000000000000000e-01
0.000000000000000000e+00 5.000000000000000000e-01
```

Figure 5.6: Packet Features

bytes) and the layer 4 header (8 or 20 bytes) are included along with either user data or zero padding. An example of the Y labels is shown in Figure 5.7. The labels are passed to the neural network for the error calculation.

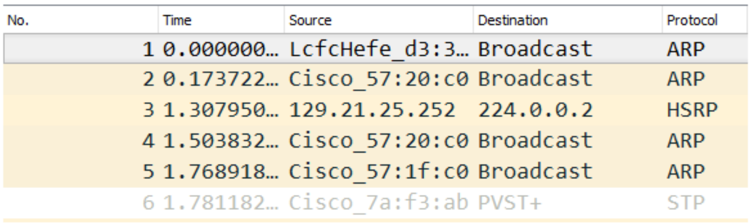


```
0.000000000000000000e+00
0.000000000000000000e+00
8.000000000000000000e+00
0.000000000000000000e+00
0.000000000000000000e+00
1.000000000000000000e+00
```

Figure 5.7: Ground Truth Labels

Based on this list, the first six packets would have been ARP, ARP, IPv4 UDP HSRP, ARP, ARP and 802.3 STP. This can be verified from the Wireshark capture as shown in Figure 5.8.

The features are input to the neural network which then completes its forward pass. At the output, the calculated values are passed to a Softmax function which provides calculated class (probability) associated with the current packet. This probability is returned and used to determine the difference (error) to the labeled class. At this point the backpropagation begins and, using gradient descent and an update function, modifies the weights used in the



No.	Time	Source	Destination	Protocol
1	0.000000…	LcfcHefe_d3:3…	Broadcast	ARP
2	0.173722…	Cisco_57:20:c0	Broadcast	ARP
3	1.307950…	129.21.25.252	224.0.0.2	HSRP
4	1.503832…	Cisco_57:20:c0	Broadcast	ARP
5	1.768918…	Cisco_57:1f:c0	Broadcast	ARP
6	1.781182…	Cisco_7a:f3:ab	PVST+	STP

Figure 5.8: Actual Packet List

linear combination calculations. With these updates, the forward propagation phase can run again. Over time, the loss function approaches a minimum or an escape value. In our case, the first network classifier had a collection of trials run for 1000, 5000, 10000, etc. iterations rather than an escape cost. Once the network is trained, it is run again against the test set and these recognition rates are reported below.

Initially, the network classifier batch processed the entire dataset making training of the neural network slow, often taking more than 20 hours. Mini-batch decreases this time significantly. For example, using batch processing and 116 input features, the network would have to run for more than 30K iterations and nearly 24 hours to achieve recognition rates of 93-94%. The current structure using mini-batch accomplishes similar rates with 5000 iterations in about 4 hours. As noted earlier, a variety of mini-batch sizes (64, 128, 256 and 512) were tested. It should be noted that many of these earlier experiments were run on a CPU. Later work was moved to GPUs.

Learning rate can drastically affect the performance of the neural network. Learning rates that are too small (very small steps in gradient descent) lead to slow training. Large steps can cause the calculations to pass over the loss function minimum. For testing a variety of learning rate values ranging from .01 to .4 were used. For most of the tests in this chapter, .01 and .0001 were used almost exclusively.

5.5 Results

In order to familiarize ourselves with the work of other research groups, we first attempt the same classification tasks with the datasets from [45]. After reducing the number of features to 22 (IP addresses, header sizes, etc.) recognition rates very close to those in [44] are achieved. The concern that the heavily weighted dataset may skew results is shared and this is another reason for not only moving to actual packet captures but newer datasets. What follows are results from the previously described preprocessing plan and the neural network configuration for the successful classification of both packets and flows.

The training set has 101106 packets and the test set has 10725. The Table 5.3 results are received after 5000 and 10000 iterations. During testing there were additional batch sizes utilized however the best experimental results were achieved using batches of 64 and 128 packets and so these are included here. The results include different preprocessing sizes. The learning rate is fixed at .01. For verification, the recognition rate for the training set was greater than 99.9%. This simply indicates that the operation and labeling, etc. are functioning as intended. The test set recognition rate varies based on the size (features) of the input. RR is an abbreviation for Recognition Rate.

Table 5.3: Accuracy - Minibatch

Features	Batch Size	5k Cost	5k RR	10k Cost	10k RR
125	64	.000134	.978	.000092	.977
150	64	.000057	.961	.000066	.96
175	64	.000054	.959	.000069	.959
200	64	.000057	.951	.0001	.9513
1444	64	.000053	.89	.000045	.895
125	128	.000252	.98	.000016	.991
150	128	.000019	.972	.000029	.981
175	128	.000049	.951	.000014	.964
200	128	.000024	.956	.000017	.969
1444	128	.000018	.894	.000007	.921

Even at this number of training iterations there are some interesting trends to note. The first is that as the number of input features increases, and especially once it increases to 1444 bytes, the recognition rate of the test set goes down. As noted earlier, packets of this size may have a considerable amount of zero padding which may have a negative impact. Perhaps a bigger obstacle is the training time with the larger packets (1444 bytes) doubling the training time of the smaller sizes. There is a smaller training time difference for 125, 150, 175 and 200 features. For example, at 125 features, the training time for 1000 iterations 58, 52, 54 and 45 minutes for 64, 128, 256 and 512 batch sizes respectively. In all cases, once the network is trained it takes approximately 3 seconds to apply the weights and effectively label the test data set - one of the advantages of the neural network.

When the number of hidden nodes was changed to reflect the number of classes, the recognition rate in the best case test set (125 features, batch size of 128 and 10k iterations) improves to 99.87% after just 5000 iterations. Table 5.4 presents a selection taken from the most significant of these results.

Table 5.4: Recognition Rates - Minibatch, 28 hidden nodes

Features	Batch Size	5k Cost	5k RR	10k Cost	10k RR
125	64	.000074	.987878	.000058	.983124
125	128	.00023	.998788	.00023	.998788
125	256	.000054	.9986	.000019	.99963
125	512	.000074	.9845	.000017	.99804

Clearly the model does well with overall packet classification rates (greater than 99%) but an important question is whether the model handles individual class. For the same dataset, a few of the main categories are more closely examined. For space, Table 5.5 only depicts the categories for UDP, TCP, ICMP, ARP and 802.3. Clearly it shows good performance with individual classes though a few packets are misclassified.

Table 5.5: Recognition Rates per class

Category	Parsed Report	NN report	Percent
UDP	1651	1628	.986
TCP	9004	9008	1.0004
ICMP	0	0	0
ARP	49	49	1
802.3	17	17	1

The results from Table 5.3 can be interpreted another way. Reorganizing the data by input features and comparing the performance of the various batch sizes, parameters that work well at a certain number of iterations can be determined. This can be seen in Figure 5.9. For clarity the results from 5000 iterations are shown. The 1444 feature set has been removed for scaling.

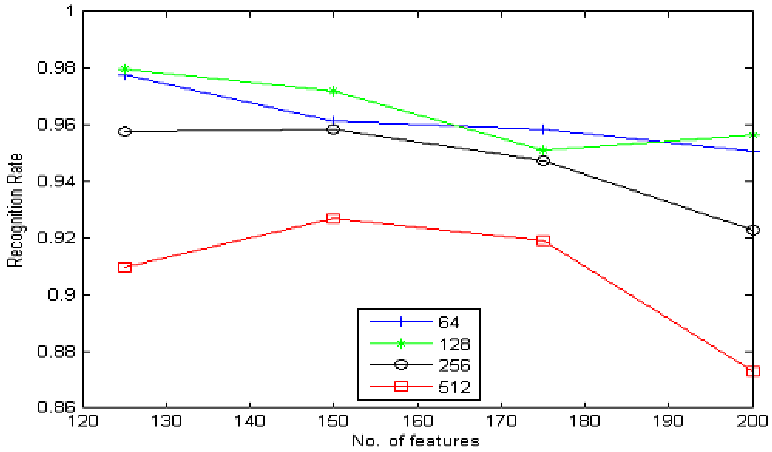


Figure 5.9: Recognition Rates

While batch size does not appear to have a large impact at this point, there is a slight improvement with a batch size of 128. These batch sizes were chosen as part of best practices but there is some indication that the best batch size may have some connection to the hardware of the machine completing the processing. While the analysis was not part of this work, the machine specifications are included for completeness. For these results, two separate Windows 10 machines were used. The laptop had 12GB of RAM and a 2.6GHz Intel Core i7-3720QM processor. The desktop had 16GB RAM and a 3.6GHz Intel Core i7-3820 processor.

As mentioned previously, tests were run against an internal structure of 20, 28 and 36 hidden nodes. This affects the number of calculations and changes the impact of a node. Table 5.6 shows these results. Like the 28 node structure, at 2000 iterations the best performance for the 20 and 36 node structures was found when the batch size was either 64 or 128. The best recognition rates at 2000 iterations for 20, 28 and 36 nodes were .989, .993 and .989 respectively. The 64 and 128 batch sizes were run again at 5000 iterations in order to compare them with the previous results. The columns indicate the number of hidden nodes and the batch size.

Table 5.6: Hidden Node Recognition Rates

Features	20-64	20-128	28-64	28-128	36-64	36-128
125	.985	.989	.978	.992	.983	.987
150	.978	.983	.978	.995	.987	.994
175	.949	.975	.971	.988	0.974	.986
200	.94	.959	.958	.964	.962	.969

One interesting, though perhaps expected, outcome is that while the training time for larger batch sizes goes down, so does the recognition rate. In addition, as the number of hidden nodes increases, so does the training time though it is not drastic. This increase is probably due to the increase in the number of calculations done for each linear combination and the propagation time.

5.5.1 Multiple datasets

Once the structure testing was complete, the best model for classifying other datasets is selected. The best results were achieved using an input feature size of 150 (75 bytes), 28 hidden nodes and a batch size of 128. The additional test datasets were chosen from a variety of packet distributions and capture sizes. Some of the results are provided in Table 5.7.

Table 5.7: Dataset Recognition Rates

Dataset	Size	Recog Rate
0	21497	.846
1-5	20-26k	.997
6	128100	.997
7	191186	.993
8	6414	.138

The recognition rates for most of the datasets exceed 99% with many exceeding 99.5%. However, included are some datasets with different accuracies. Dataset 0 was heavily weighted towards some of the lesser utilized protocols and so this recognition rate is likely due to insufficient training for these categories. Dataset 8 is a much smaller set of packets and comprised mostly of

Real Time Transport traffic which is not a classification category. It appears that the port numbers used varied and so the traffic may have been misclassified. These outcomes demonstrate that a neural network structured in this way and provided correctly preprocessed files can successfully classify a wide variety of contemporary traffic types but a strategy should be developed to handle unidentified classes.

5.5.2 Traffic Flows

When processing flows, the data is limited to a select set of bytes. Flows are uni-directional and defined by source and destination addresses, ports and protocol identification fields. Specifically the source and destination MAC addresses, source and destination IP addresses, layer 2 type field, layer 3 protocol ID field and the source and destination ports are used. If a particular field is not present (ex. ICMP and port numbers) it is set to 0. This collection of fields also makes the number of features input to the neural network fixed at 27 bytes or 54 hex characters. The downside is that some of the detail that makes packet classification so successful is lost. There are number of other factors that make flow classification based on these fields challenging. Applications using the same port numbers (ex. port 80 or 443) or traffic that does not use the port number at all can make identification difficult.

For tests using a similar structure, the recognition rate for flows quickly reaches 88.1% regardless of changes to input features or the neural network structure. However, several tests resulted in recognition rates reaching 99.8% A review of the datasets reveals that overfitting skewed the test. It may also be that with this number of features, the model may not generalize as well as the previous cases or that the missing data (at least 71 hexadecimal characters) is critically important.

An example can be found in the length field. In some cases, packet length can be very telling as in a file transfer that uses full size frames. However, telnet or SSH streams do not. In addition, flow classification can be challenging when the number of packets in the flow is small. This question becomes important when messaging such as DNS or ARP is considered. Clearly there is a conversation, but are these really flows? Discounting all of these conversations cannot be done without considering the percentage of traffic that might be lost as discarded micro-flows can provide insight into the other traffic.

5.6 Chapter Conclusion and Contributions

Traffic classification is an increasingly important component for communication network visibility. It is our belief that raw network traffic should be used where possible because it is more representative of actual conditions. The chapter demonstrates that machine learning techniques such as neural networks are not only able to process this data in a timely and accurate manner but at very high accuracy rates. This chapter presents the preprocessing techniques and feature selection for network traffic and the neural network structure used classify both packets and flows, including the hyper-parameters used. Also shown were the results of principal component analysis which does not provide an automatic improvement in training time or classification rates, at least with the datasets used here.

After thorough study, the best results were achieved using an input of 150 features (75 bytes), batch size of 128 and a hidden nodes equaling the number of output classes. The model can successfully classify network traffic in the aggregate above 99% and individual classes above 98.6%. By changing the neural network structure the model achieves greater than 99% over a variety of datasets. In our more recent work network flows are classified with an accuracy range from 88% to 99%. This work is ongoing. Network traffic classification is a task that is part of a larger whole. Next steps will include improvement to flow recognition rates, applications such as security, and combining this work with software defined networking and inference architectures.

Chapter 6

Optimization and Balancing

6.1 Operational Background

This chapter provides some detail regarding the structure of the neural network used in this work and the order of operation including the optimizers. Much of the architecture is based on the authors previous work in this space and has been included in chapters 4 and 5.

Building a multi-layer perceptron (MLP) classifier requires that close attention is paid to the data at all stages of processing. This includes preprocessing/parsing, normalization, structuring of the network layers, organization of the input, loss calculation and techniques used in the backpropagation phase. During these steps choices are made regarding a collection of available algorithms and structures in order to determine what works best for a particular task. This chapter seeks, among other things, to help make these decisions a little more straight-forward.

Normalization is the first step process of scaling all the vectors to similar, small dimensions around the origin. Processes such as Principal Component Analysis work best when the data is "mean normalized" which subtracts the mean from each value and divides by the variance: (x - μ)/σ^2. Normalized data is then processed by the artificial neural network structure. In this case the datasets are comprised of packet samples and a desired number of bytes (features) from each. Thus input features are part of the raw packets flowing across the routed infrastructure. The resulting m x n matrix is the number of packets x the number of selected features. It was found that collecting the first

160 hexadecimal characters or 80 bytes of each packet maximizes accuracy. For this work the physical and protocol addresses of each packet are zeroed.

The base neural network structure with an input layer, two hidden layers and the output layer is shown in Figure 6.1. The layers are comprised of nodes.

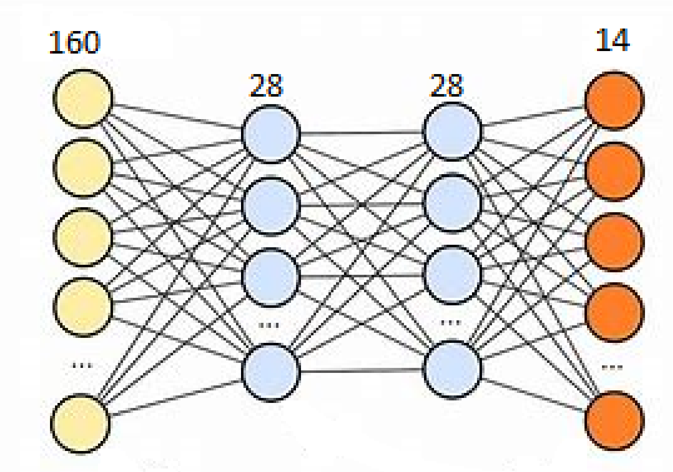


Figure 6.1: Neural Network

The hidden layers vary in size based on the experiment but the input size of 160 and the output size of 14 classes are fixed. Moving through the layers, nodes represent linear combinations of the previous layer and these are passed through an activation function. Each node is directly connected to every other node in the preceding and subsequent layers. These are also called fully connected layers. Thus, each node has a calculated value and each connection has a weight. These values (X) and weights (W) are stored in matrices; $X = [x_0, x_1, ..., x_n]$,$W = [w_0, w_1, ..., w_n]$ and the dot product result is $Z = [z_0, z_1, ..., z_n]$. This result is passed through an activation function. Activations include the sigmoid, tanh and ReLU (rectified linear unit) functions. Thus the hypothesis calculation becomes:

$$h_0 = g(x_0w_0 + ... + x_nw_n) = \sum_{i=0}^{n} x_iw_i$$

where g is the activation function, X is the input vector and W is the weight vector. Each layer performs these calculations until the last layer where an error for the correct label is established via a loss function. In the last layer these values are passed to a function called Softmax. The Softmax function determines the probability for each label. This is compared to the ground truth for the loss calculation. Once the loss drops to an acceptable level or the specified number of iterations has run, the label is assigned.

At this point, forward propagation through the neural network is complete and now the error is backward propagated through the network and the weights used in the forward calculations are updated. This process is called gradient descent (GD) which uses a learning rate and partial derivatives to take a step in the direction of the loss function minimum. Additional details regarding this operation can be found in Chapter 4.

Neural networks differ in many ways (hyper-parameters, structure, choice of features, etc.) even if they are working on the same problem. Models are often evaluated based on their accuracy, stability and especially training time. A key component is the behavior of the neural network during the backpropagation phase. To address these an optimizer is inserted into this process after the layered model runs. Techniques such as momentum (exponentially weighted averages) and the associated bias correction have been developed that further reduce training time and increase accuracy. The discussion of these techniques can be found in chapter 2.

6.2 Datasets

As shared previously, this work is concerned with the classification of packets from a standard communication network. This traffic is obtained from a locally constructed testbed. Live capture from a collection of Windows hosts and Linux VMs gathered the packets as they flowed across the interconnected routers and switches. The testbed structure and some previous results can be found in [29] and [27]. In these works, campus and testbed network traffic was monitored in order to determine the most likely set of protocols that would be seen on a general network. This resulted in an initial set of classes and large training sets of varying sizes.

For this work, the use of completely balanced training datasets was a goal. This type of dataset is more difficult and time consuming to construct and so the number of classes was reduced to thirteen. In order to achieve satisfactory accuracy rates, each class in the balanced trainer had to have 5K samples. The final classes types (loopback, CDP, STP, ARP, ICMP, IGMP, UDP SSDP, UDP DNS, UDP DHCP, UDP NBNS, TCP port 80, TCP port 8080 and TCP port 443) were included to represent different communication layers. The resulting training dataset had 65K samples. A balanced validation set was also created that contains five hundred samples from each class for a total of 6500 samples. Lastly, four unbalanced datasets from additional live captures are included. The only limitation for the datasets is that the packet classes included must be from the protocol list.

Balancing packet traffic datasets is difficult because even variations in protocol versions or message type can cause misclassification errors. For example, the spanning tree protocol uses a collection of messages between network switches to ensure loop free connectivity. However, spanning tree variations (802.1D, 802.1Q Rapid, PVST) are different enough that when mixing versions 100% of the packets in the test dataset were classified as a different protocol entirely. Another example can be found in ICMP information vs. ICMP error messages as the latter contain 64 bits of the original message. Thus future work will include a much more specific set of classes. Chapter 3 contains a full description of the challenges and solutions associated with packet traffic datasets.

6.3 Optimization

There are several optimization techniques available in most machine learning packages. Investigations here included Adadelta, Adagrad, Adam (SparseAdam), Adamax, Averaged Stochastic Gradient Descent (ASGD), Limited Broyden Fletcher Goldfarb Shanno (L-BFGS), RMSprop, Rprop and Stochastic Gradient Descent (SGD). While many of these have been in existence for some time, several have emerged more recently. The experimental results shown here were achieved using the pytorch optim package but apply across a broad range of environments. This brief explanation is arranged alphabetically and is a synopsis of the descriptions from Chapter2.

Adagrad [18] is an optimization technique that examines the geometry of the data features and in this way may be able to leverage important yet infrequently occurring details. The authors go so far as to state that it is "silly" to have a global learning rate rather than one for each feature. In our case, one concern was that packet traffic offers many features that have little impact but high variance.

Derived from Adagrad, Adadelta [76] is described as an adaptive gradient descent method in that the per-dimension learning rates adjust. This work also reminds us that selecting hyper-parameters such as learning rate can be a bit of an art form. The authors consider the drawback of Adagrad (decay of learning rate, manual global learning rate) as they build in automatic adjustment.

Adam (also SparseAdam) [38] is an algorithm that leverages the benefits of two other algorithms (Adagrad and RMSprop). This adaptive moment estimation method updates exponential moving averages gradient and the rate of decay is controlled by other hyper-parameters. Adam is arguably one of the more popular and perhaps "default" optimizers used today.

Adamax is a variant of Adam that addresses instability problems in the update rule for the weights in the presence of large parameter values. This occurs when the L^p norm is used to generalize the L^2 norm.

Averaged Stochastic Gradient Descent (ASGD) [51] is one of the more mature ideas in the library and leverages matrix value estimation and averaging to reach optimums. The authors of [51] shared that at the time of that work, this might have been necessary because the optimal algorithm could not be implemented.

An implementation of the Limited Broyden Fletcher Goldfarb Shanno (L-BFGS) method can be found in [7]. The problem of inefficient gradient descent and meta-optimization optimization (hyper parameter tuning) is addressed by measuring the energy of the descent step and automatically updating the direction and step used.

The work done with RMSprop is found in a lecture by Geoff Hinton [33]. It is suggested that the learning rates be adjusted at stages during training in order to improve mini-batch gradient descent performance. RMSprop adjusts the learning rate by dividing it by an average of the recent gradients. Other improvements include the use of momentum. The efficacy of momentum can be found in [67]. A Nesterov modification [46] to momentum is also included in the pytorch implementation.

The Rprop (resilient propagation) approach [56] examines the behavior of the loss or error function. The concern then is not for the gradient itself but the sign of the partial derivative as a change indicates a step too far in the descent.

Stochastic Gradient Descent (SGD) (4.4) has the momentum and Nesterov options.

6.4 Methodology and Initial Results

Optimization techniques modify the behavior of the neural network, particularly the weight updates. However, not all optimization algorithms generalize well. In this case, packet traffic classification presents a unique problem in that features that may have very little to do with the classification task often have the highest variance. For example, MAC or physical addresses are large six byte fields that are different for every single node. For unicast traffic, this expands to twelve bytes because both the source and destination address must be considered. However, the addresses may help identify some traffic it may be broadcast or multi-cast. This same discussion can be extended to Internet Protocol (IP) addresses.

One of the motivations for this work is that techniques such as principal component analysis (PCA) [72] do not seem to offer the same results as seen in other classification tasks [29]. For this reason, all of these investigations are done on datasets with the layer 2 and layer 3 addressing set to zero. Another concern is that important features and the location of these features in each sample can vary even if the packet headers are somewhat fixed in their content.

To determine the best methods for this problem and to understand the pytorch implementation, we deploy a multi-layer perceptron neural network containing two hidden layers. The input layer takes packets that are divided into thirteen classes and selects 160 features (80 bytes) from each sample to ensure that header fields are included. For the initial tests, default settings (no weight decay, no momentum, no helper algorithms) are used. The only deviation is when a learning rate does not allow the model to converge. In those instances the learning rate was typically set to 1e-3. Many of the optional settings (ex. momentum) are considered important improvements to some algorithms and so it is somewhat surprising to run these initial experiments without them.

All of the datasets are run through the neural network three times and the average accuracies are recorded here. The initial tests are set to run for either 2000 iterations or stop when reaching the initial escape loss of 1E-20. The low loss rate value allows the operation of each technique to be observed as it is rarely reached. Subsequent tests will be modified based on these results and some other available options. This primary goal at this point is to experiment with the base technique and then, where appropriate, add momentum, weight-decay and any add on algorithms. There are four identical physical machines used for the trials and all of them use an Nvidia Quadro P1000 GPU.

For clarity, Table 6.1 provides a description for all of the tables that follow. The abbreviations R (regularization/weight decay), M (momentum) and N (Nesterov) are used throughout. To start, all of the algorithms are deployed with the default settings and a loss of 1e-20, which provides an idea of the training time and accuracy. This also reveals what loss rates can be expected from each.

Table 6.1: Experimentation

Table	Algorithms	Escape Loss
2-Initial	All	1e-20
3-Momentum etc	RMN Capable	1e-8
4-Initial	Best 5, 97% Acc	1e-5
5-Momentum etc.	Best 8	1e-5
6-Momentum etc.	Best 8	1e-3
7 Bal vs Unbal	Best from all	1e-5, 1e-3

In Table 6.3, momentum, regularization, etc. are activated but against the experimentally determined loss rate of 1e-8. Subsequent experimentation had the primary objectives of improving accuracy or reducing training time for the best performing optimization techniques. Tables 6.11 and beyond target improving accuracy via neural network structure changes and examine the impact on training time.

Thus the contributions of this chapter include not only the investigation into optimization techniques and network parameters, but an investigation into wider and deeper networks for this application.

For SGD, results with an N indicate the Nesterov algorithm. [33] states that Nesterov is superior choice when possible and so it is added to this round. The network configuration is fixed at two hidden layers and 28 hidden nodes per layer. The first result does not have an optimizer and uses a learning rate of 1e-6 which based on our previous work. Optimizers often have a recommended value for learning rate. When convergence is not achieved with the default settings, the learning rate is set to 1e-3 or 1e-2. These are noted in the affected tables.

Table 6.2: Optimization Initial Results

Method	Training (min)	Ave Acc %	Std Dev
None	1	88.2	.087
AdaDelta	46	98.4	.013
AdaGrad	43	98.2	.013
Adam	46	98.0	.025
Adamax	50	98.7	.010
ASGD	43	97.9	.014
L-BFGS	53	45.1	.291
RMSprop	47	89.2	.194
Rprop	64	10.98	.171
SGD w/o N	43	86.0	.193
SGD w/ N	43	66.7	.383

As can be seen, the results vary widely. All of the algorithms took in excess of 40 minutes to complete 2000 iterations though this was with an escape loss of 1e-20. Running without an optimizer results in accuracies lower than 90%. For average accuracy, AdaDelta, AdaGrad, Adam and ASGD performed above or near 98%. Both RMSprop and SGD had acceptable performance until some results came back below 50%. Because this happened more than once during testing, they are not considered outliers and are included in the average. This is reflected in the test dataset accuracy standard deviation for both in that the values are much higher than the other algorithms. SparseAdam was also tested but the tensor/gradient was too dense for the algorithm. Rprop and L-BFGS had the worst performance with widely varying trials and low accuracy rates. They also claim the longest average training times.

It must be remembered that all of these results come from the default settings for each method though AdaGrad, ASGD, RMSprop and Rprop had their learning rates adjusted to 1e-3 from a base of 1e-2. This change was made after initial tests failed to converge.

6.5 Weight Decay and Momentum

The large array of options for all of the optimization techniques makes comparing them difficult and so this section will be limited to the exploration of momentum, weight decay, centering (variance) and where appropriate, "helper" algorithms such as Nesterov and AMSgrad. For example, Stochastic gradient descent (SGD) has momentum in use by default but the Nesterov algorithm and weight decay are not. It should be understood that several of the optimization techniques have additional options that are not examined in this paper.

Because the optimization methods differ in their useful learning rates and additional parameters, the goal of this section was to improve on the default settings using weight decay and/or momentum for each rather than declare a "best" - though in many cases this was the eventual outcome. It may also be the case that including weight decay, momentum, etc. slows training or increases loss. Table 6.3 records the results from activating these. Common values for weight decay (regularization) and momentum are 1e-2 and .9 respectively and so these values were used throughout these tests. LBFGS and Rprop do not use either and so are not included here.

One other aspect that is modified for this section is the escape loss value. In our previous work we have found that results at or near 99% accuracy can be reached with a loss or error of 1e-5. The algorithms studied here reach error rates ranging from 1.8e-5 to 8.1e-15. The mean and median values are 6.2e-4 and 4.9e-8 respectively. For this set of tests, the model is allowed to terminate if a loss of 1e-8 is reached. This value helps to determine if an algorithm achieves an improvement in speed and accuracy.

The Adam algorithm also has the option of using an AMSGrad variant [53] so this trial is included. The RMSGrad version includes knowledge of past gradients to help ensure convergence. Table 6.3 also makes room for a few other options. For example RMSprop utilizes momentum (M), weight-decay/regularization (R) and centering based on the variance (V).

Table 6.3: Weight Decay and Momentum Results

Method	Training (min)	Ave Acc %	Change	Std Dev
AdaDelta R	37	98.25	-.15%	.01
AdaGrad R	53	97.83	-.37%	.018
Adam R	49	98.3	+.3%	.01
Adam AMS	49	98.23	+.23%	.01
Adamax R	49	98.72	+.02%	.1
ASGD R	49	97.85	-.05%	.026
RMSprop RM	50	75.66	-22.24%	.223
RMSprop RMV	48	72.06	-25.84%	.273
SGD RM	43	98.85	+12.85%	.009
SGD RMN	43	98.8	+32.1%	.011

Even with weight decay (regularization) and/or momentum, all of the algorithms completed 2000 iterations without hitting the escape value. In fact, the lowest observed loss was 4.5e-6. The training time changes little though exceptions can be found in AdaDelta and AdaGrad. For accuracy there is a massive improvement for SGD of 12.85% without Nesterov and 32.1% with Nesterov. RMSprop suffers a 20% decline in accuracy. Like the previous results, the algorithms that do not perform well also have the largest standard deviations for accuracy. It should be noted that Adamax did have a single outlier which was removed for these calculations and improves its accuracy.

6.6 Improving Training Time

With many of the algorithms returning prediction accuracy better than 98%, a reasonable question to ask is whether the training times can be reduced. Training time is a function of training set size, model architecture and hyperparameters such as the learning rate. Changing all of these factors would make it difficult to compare results with the previous test outcomes. So, for this set of trials the best configurations from both sets were selected and the escape value is changed to 1e-5. In this way we hope to determine what algorithm might still provide acceptable accuracy while cutting training time.

Tables 6.4 and 6.5 depict the results for each category. A value of 97% accuracy was chosen as a cutoff to include as many of the good performers as possible. In Table 6.4 AdaDelta, AdaGrad, Adam, Adamax and ASGD were all run again using the default values (no weight decay, momentum, etc.) against the escape loss of 1e-5. All five had a significant reduction in training time with three of them finishing in less than one minute. Even with this remarkable decline in average training time, all except AdaDelta approached or exceeded 98% prediction accuracy with Adam topping the list at 98.59%. AdaDelta shows a distinct loss in performance however there were a pair of outliers during testing. Without these, the AdaDelta accuracy would be 96.4%. In Table 6.5, AdaDelta, AdaGrad, Adam, Adamax, ASGD and SGD were also run again using combinations of momentum, weight decay or Nesterov against the 1e-5 escape loss value.

Table 6.4: 1e-5 Results no RMN

Method	Train Time	Ave Acc %	Change	Std Dev
AdaDelta	3s	88.92	-9.48%	.193
AdaGrad	7min	97.87	-.33%	.012
Adam	24s	98.59	+.59%	.009
Adamax	26s	98.44	-.28%	.009
ASGD	5min	98.18	+.28%	.01

Table 6.5: 1e-5 Results for RMN

Method	Train Time	Ave Acc %	Std Dev
AdaDelta R	58	98.28	.013
AdaGrad R	44	93.57	.145
Adam R	45	98.77	.01
Adam R+AMS	55	98.64	.012
Adamax R	62	98.49	.013
ASGD R	54	98.7	.011
SGD RM	55	98.57	.013
SGD RMN	54	98.82	.009

An important note on these results is that if an algorithm did not reach the escape loss value of 1e-5, the model completed 2000 iterations. Thus, the trial becomes a repeat of the results shown in 6.3. It turns out that all of the algorithms selected to run again did not reach the loss value of 1e-5. This is a little surprising given the goals of momentum and weight decay. It is possible that if the trials were allowed to run until the loss value was reached, prediction accuracy might be improved however many of the algorithms simply did not show signs of reducing the loss to that level. It may also be related to the challenges in working with packet traffic datasets. While these results are included for completeness, it was decided that another series using an escape loss value of 1e-3 should be run. These results are shown in Table 6.6.

Table 6.6: 1e-3 Results for RMN

Method	Train Time	Iterations	Ave Acc %	Std Dev
AdaDelta R	35min	1419	98.5	.012
AdaGrad R	21s	15	98.07	.012
Adam R	2min	88	98.79	.01
Adam R+AMS	27min	1161	98.54	.013
Adamax R	49min	2000	98.49	.013
ASGD R	21s	15	97.91	.017
SGD RM	14min	668	87.55	.194
SGD RMN	2s	1	90.98	.149

Once the escape loss value was adjusted the training time scale for several of the techniques shifted from minutes to seconds with AdaGrad, Adam with Weight Decay, ASGD and SGD with Nesterov finishing under 120s on average. More importantly, AdaGrad, Adam and ASGD also maintained predication accuracies at or near 98%. They also have very low accuracy standard deviations. Thus, it would appear that these and some of the algorithm configurations seen in 6.4 would be good choices for neural networks targeted at packet traffic or similar datasets.

6.7 Balanced vs. Unbalanced

A big part of the work done for this chapter was in building balanced datasets as described in [10]. This was in part due to what the authors describe but also because some of our own work, while highly successful, would result in occasional drops in prediction accuracy between test sets. In this section we explore what happens if the training set has the exact same classes included but the number of samples in each is not balanced. These results are shown side by side in Table 6.7.

Table 6.7: Balanced vs. Unbalanced

	Balanced		Unbalanced	
Method	Time	Accuracy	Time	Accuracy
AdaGrad	7min	97.87	3min	94.84
Adam	24s	98.59	18s	97.33
Adamax	26s	98.44	27s	93.39
ASGD	5min	98.18	88s	95.9
AdaDelta R	35min	98.5	8s	97.77
AdaGrad R	21s	98.07	14s	92.93
Adam R	2min	98.79	8s	96.46
Adam R+AMS	27min	98.54	12s	94.49
Adamax R	49min	98.49	10s	94.37
ASGD R	21s	97.91	12s	94.07

What is revealed in these results is that almost without exception, the unbalanced datasets reduce training time by some small amount. However, there is a significant cost to prediction accuracy. Every single optimization technique loses at least .73% with most losing much more. For example Adamax and AdaGrad lose more than 5%. Clearly, datasets, especially those dealing with packetized traffic or similar datasets must be balanced. AdaDelta without weight decay (default), SGD (with weight decay) and SGD (with weight decay and Nesterov) were dropped due to having more than 1 outlier or poor previous performance.

6.8 Improving Accuracy

After experimentation with the optimization algorithms, a reasonable question is whether, given the constraints of the previous testing, the accuracy can be improved. The goal is to move the average accuracy closer to the near Bayesian accuracy seen in the trainer. The included optimization algorithms are chosen based on the best previous results from Tables 6.4 and 6.6. Based on the last set of results for the best performers, the overall accuracy average is about 98.34%. Thus, the margin for improvement is between 1.6% and 1.7%.

As the performance of the model is already quite good, it appears that the task is to adjust for the variance or perhaps noise in the data. This is typically done using more data, model architecture changes or regularization. However, the datasets will not be modified and so the investigation is limited to architecture modifications and regularization options noted earlier.

In an effort to identify possible trends or opportunities, an experimentation matrix was constructed and this is shown in Table 6.8. Both the hidden nodes and layers are doubled for several steps. Additional tests using 84 hidden nodes were added due to good performance discovered in a random test. The "X" indicates tests completed. After careful consideration of the training time constraints, certain experiments were strategically chosen to explore all cases of interest while avoiding configurations known to under-perform or add little value.

Table 6.8: Test Matrix: Hidden Nodes vs. Layers

Hidden Nodes	1	2	4	8
14	X	X		
28	X	X	X	X
56	X	X		
84	X	X		
112	X		X	
224	X			X

6.8.1 Wider Networks

The approach of using hidden nodes ranging from the number of classes (14) to wider networks (up to 224 hidden nodes) and increasing the number of layers is informed by the recent work [59]. The baseline results shown in Table 6.9 are from a single layer network that varies the number of hidden nodes.
From Table 6.9 it can be determined that the best hidden node configuration is either 28, 56 or 112 as these all approach 98% accuracy on average. The best performers in terms of algorithm are Adam +R+AMS and default Adam. AdaGrad, Adamax, ASGD and Adamax +R all approach 98% as well. If the performance at 14 hidden nodes is ignored as an outlier, then all but AdaDelta, AdaDelta +R and Adam +R exceed 98%.

Table 6.9: Improving Accuracy: Single Layer

Method	14	28	56	84	112	224
AdaGrad	.9329	.985	.9865	.9867	.9862	.9871
Adam	.938	.9854	.9874	.9883	.9878	.9873
Adamax	.936	.9858	.988	.9881	.9878	.9876
AdaDelta	.9356	.986	.9384	.8922	.983	.8828
ASGD	.9785	.9841	.9858	.9392	.9863	.9867
AdaDelta +R	.8921	.9872	.9385	.9751	.89	.9339
AdaGrad +R	.9324	.976	.9846	.9846	.9836	.9838
Adam +R	.9352	.938	.9369	.8881	.9356	.8934
Adam +R+AMS	.9848	.9862	.9875	.9876	.9877	.9879
Adamax +R	.9345	.9854	.986	.9848	.9864	.9866
ASGD +R	.8732	.9808	.9837	.985	.9834	.9857
AVE	.9339	.9799	.9729	.9636	.9725	.9639

As one might expect, training time reveals that as the number of hidden nodes increases, training time trends up though not significantly. The more important detail is the time scale for the various techniques. Table 6.10 provides this extra information.

Table 6.10: Time Scale

Method	below 1 min	1-30 min	above 30 min
AdaGrad		X	
Adam	X		
Adamax	X		
AdaDelta	X		
ASGD		X	
AdaDelta +R	X		
AdaGrad +R	X		
Adam +R	X		
Adam +R+AMS		X	
Adamax +R			X
ASGD +R	X		

After analyzing the information contained in Tables 6.9 and 6.10, it can be concluded that in terms of time and accuracy, default Adam, default Adamax and Adagrad +R would be good choices. Recall that the default configurations do not utilize regularization or weight decay.

6.8.2 Deeper Networks

Similar tests are performed after adding a second layer with a focus on the configurations that performed well in terms of training time and accuracy. Table 6.11 provides the result of having this second layer with 28, 56 or 112 hidden nodes in each of the hidden layers.

Table 6.11: Improving Accuracy: Two Layers

Method	28	56	112
Adam	.9864	.9846	.9857
Adamax	.9828	.9853	.987
AdaDelta	.9814	.9765	.8765
AdaDelta +R	.9867	.986	.9364
AdaGrad +R	.9759	.9815	.9823
Adam +R	.9843	.877	.9685
Average	.9843	.877	.9685

In this configuration the performance for default Adam and default Adamax exceeds 98.5% with AdaGrad +R achieving 97.99%. The training time for AdaDelta +R jumps to nearly an hour while the rest remain under a minute to reach their assigned escape loss values.

Continuing the exploration of deeper networks, the high performing configuration of 28 hidden nodes is used. The results for the same algorithms over 1, 2, 4 and 8 layers is shown in Table 6.12.
From this data it is clear that moving to deeper networks does not provide any benefit for these datasets. Depending on the algorithm, one or two layers is best. At eight layers the performance is abysmal regardless of the optimization used. In addition, at eight layers default Adam, Adamax, AdaDelta often do not converge. With regularization, all of them were able to converge on the escape loss value though the training time for AdaDelta +R was over an hour in both four and eight layer models.

Table 6.12: Improving Accuracy: Deeper Layers

Method	1	2	4	8
Adam	.9854	.9864	.9393	.8624
Adamax	.9858	.9828	.9661	.8736
AdaDelta	.986	.9814	.9678	.8101
AdaDelta +R	.9872	.9867	.9828	.9048
AdaGrad +R	.976	.9759	.8995	.9226
Adam +R	.938	.9843	.9819	.9185
Average	.9764	.9832	.9646	.882

Applying the same time scale analysis to the deeper networks, it can be seen in Table 6.13 that many of the optimizers stay below a minute in training time. For Adamax, AdaDelta, AdaDelta +R and Adam +R, the models resulted in significantly longer training times, often exceeding an hour. However, these only occurred in the deeper network architectures of 4 or 8 layers. For architectures of 1 or 2 layers, the training time was still less than one minute.

Table 6.13: Time Scale - Deeper Networks

Method	below 1 min	1-30 min	above 30 min
Adam	X		
Adamax			X
AdaDelta			X
AdaDelta +R			X
AdaGrad +R	X		
Adam +R		X	

It is reasonable to conclude that for models consisting of two layers and a hidden node configurations having number of nodes = 2 * (number of output classes), all of the algorithms listed in Table 6.12 would be acceptable choices with default Adam and AdaDelta +R achieving the highest overall accuracy.

One last question regarding the width and depth of the networks used for packet traffic datasets is the performance with both wide and deep configurations. For example, does a network of eight layers and 224 hidden nodes provide any benefit? The simple answer is no. The behavior for the best performers in terms of training time and accuracy mirrors seen in Tables 6.9 and 6.12 shows that accuracy trends down as the layers extend beyond 56 hidden nodes and as the hidden layers go beyond two. In many cases, a configuration of 224 hidden nodes and eight layers has the same lack of convergence problem or very long training times.

6.9 Discussion

Packet traffic is sometimes challenging to work with because fields with the greatest variation may have the least impact for differentiation between classes. Thus, the main motivation for this work was to address some of the questions that exist when attempting to apply neural networks to packet traffic classification. Some understanding of the base structures and hyper-parameters are part of the literature but much less is known about the proper optimization algorithms and a systematic examination regarding the impact of using wider or deeper networks has not been done. It is common to see researchers state that a particular algorithm was used but little reasoning as to why it was chosen. Importantly, very little had been done regarding the importance of balanced datasets of this type.

There are myriad configurations that might be deployed and each of the optimization algorithms has a collection of options. For this reason, our exploration was limited to neural network width and depth structure changes and the deployment of regularization, momentum and any "helper" algorithms such as Nesterov and AMSgrad. Learning rates were fixed at either 1e-3 or 2e-3 (Adamax). Initial tests used very small escape loss values (1e-20) in order to observe algorithm behavior. It was found that several of optimizers work well with the default settings (within the pytorch optim package) though the training times can be long. Other optimizers work well when weight decay and momentum are enabled. But again, training times can be long even when we raise the initial desired loss to 1e-8.

Most of the algorithms tested were able to reach prediction accuracy at or near 98%. However, one goal was to watch performance as the training time was reduced. At this point the desired loss was 1e-5 followed by 1e-3. The data reveals that a collection of algorithms can maintain high prediction accuracy even with these changes. Subsequent tests fixed the escape loss value at 1e-5 for default configurations and 1e-3 for those adding regularization, etc. This was done for the simple goal of reducing error but with the understanding that several of the optimizers could not converge on very low values with these datasets.

A good question at this point is whether all of the experiments could be run using an escape loss value 1e-3. This is reasonable because an optimizer with a default configuration might be eliminated if the training time is too long. Thus, for completeness the results for the high performing default optimizers with an escape loss value of 1e-3 are include in Table 6.14.

Table 6.14: 1e-3 Results no RMN

	1e-5			1e-3		
Method	Time	Acc	Std Dev	Time	Acc	Std Dev
AdaDelta	3s	88.92	.193	2s	.9672	.033
AdaGrad	7min	97.87	.012	28s	.9669	.033
Adam	24s	98.59	.009	3s	9797	.014
Adamax	26s	98.44	.009	15s	.9811	.011
ASGD	5min	98.18	.01	12s	.9292	.143

As might be expected, with an escape loss value of 1e-3, training times decreased for all. However, with the exception of AdaDelta (which had a number of outliers previously), all of the optimizers suffer in terms of accuracy and the standard deviation seen in test set accuracy. Thus, using a loss of 1e-5 is superior.

Another very important point was the type of data and whether or not the datasets were balanced. The data consists of packetized network traffic that includes, among other things, wide variations in payload and addressing fields. This makes classification tasks challenging. A majority of the evaluation was done using the newly constructed balanced datasets consisting of thirteen classes with 5K samples for each class. Upon successful completion of the optimizer evaluation, the balanced training dataset is switched to an unbalanced set that contained the same classes. Using the unbalanced trainer results in a reduction in training time but also a substantial decrease in prediction accuracy.

Lastly, an investigation into whether changing the network structure can have an impact on packetized traffic datasets is completed. The width of the network is varied from 14 to 224 hidden nodes and the depth changes from one to eight hidden layers. It is discovered that most of the optimizers work best with only a few additional layers and a width that is double the number of output classes. With these configurations, several of the optimizers perform well with training times remaining less than 1 minute and accuracy rates exceeding 98%.

6.10 Chapter Conclusion and Contributions

This work aims at addressing some currently open questions for the application of neural networks to packet traffic classification. A comprehensive collection of optimizers were explored with completely balanced datasets of packet traffic samples. Also examined were structures suitable for research such as this, especially wide vs. deep networks that maintain high accuracies and low training times.

It is demonstrated that balanced datasets are essential for maximizing classification accuracy and that several optimization algorithms are a good fit for the task. Most notably, Adam, Adamax, AdaGrad and AdaDelta perform well and often do so without requiring weight decay. It is also clear that for this task, deep or wide networks (or their combination) are more of a hindrance than help, resulting in longer training times and reduced accuracy.

This research shows that using small networks of two hidden layers and hidden nodes equal to twice the number of output classes, training times can be reduced to less than one minute and still maintain accuracies above 98.5%. Researchers working within the communication traffic space can now build successful neural network architectures and have an excellent basis for choosing effective optimizers and hyper-parameters.

Chapter 7

Using Neural Networks to Address Port Scans

7.1 Structure for Detecting Port Scans

Using a single neural network to handle all the necessary packet classes and the associated training set is cumbersome. This chapter concerns itself with expanding the use of neural networks for applications such as security. The focus of will be on the constant problem of TCP port scans. Like all communication issues, TCP port scans are a subset of overall packet traffic. The approach investigated separates traffic into broad types and then targets the specific protocol in question. Thus two sequential Multi-layer Perceptron Neural Networks (MLP NNs) were deployed; the first classifies network packets into general packet categories such as IPv6, IPv4, ICMP, TCP, UDP, etc. and the second NN takes the discovered TCP packets, classifies them, determines flag combinations and the traffic source.

An input feature vector of the first 150 hexadecimal characters from each packet performs well for general classification. This value is used for both stages. Currently the general classifier has 13 general classes or labels. To take advantage of wider hidden layers, the number of hidden nodes is 64. This based on an early attempt to utilize a binary labeling scheme. The first stage NN layers can be described as having 150:64:64:64 nodes. Each layer is fully connected or meshed to the next layer.

Details regarding structure and operation of the MLP NNs can be found in

Chapter 4. In addition to structure, the success of any neural network is often tied to the hyper-parameters (learning rate, batch size) and the quality of the training data. A variety of these values were tested with a learning rate of 5E-6 and batch sizes of 128-512 providing the best results. Lastly, the optimizer used is Adam [38]. It should be noted that if reading this document from cover to cover, this would appear to be a backward step. However, the investigation documented in this chapter occurred before the optimization and balanced dataset investigation.

Each of the sequential neural networks has its own training dataset. In the first stage, the general classifier is trained and then given a series of test datasets to build the predictions. The ground truth is established by a parsing program that is capable of handling a variety of input capture files including Wireshark, tcpdump and Omnipeek. This is verified manually.

In order to classify the various TCP message types, a second neural network is deployed. The first multilayer perceptron (MLP) NN only feeds TCP packets into the second. Subsequently these TCP packets are run through the TCP classifier model. The most significant difference between the two networks is that the TCP neural network has 23 classes and 23 hidden nodes in the hidden layers and so might be described as 150:23:23:23. The architecture depicting the NN inter-connection is shown in Figure 7.1.

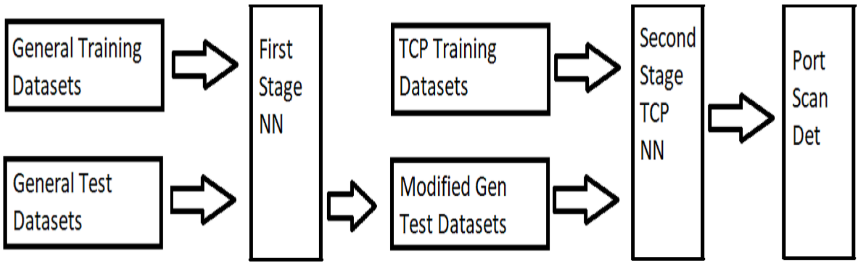


Figure 7.1: Model Architecture

As the first classifies network packets into general packet categories, the index value of each sample is recorded. After each test dataset is processed, the TCP packets are pulled out. Since the TCP classification depends on this stage, it is vitally important that the general classification is correct. We evaluate input features of 132-150 hex characters (66-75 bytes). Larger input vectors would include a TCP header with options. The learning rate is varied over .000001-.000005 based on previous work [29] and experimentation with the added datasets. A typical escape loss value after 2000 iterations would be 1E-10.

The TCP MLP uses a specialized TCP training set as the difference between packet types is often the twelve bit TCP flags field and the direction (internal and external) of traffic. Once this has completed, the TCP test datasets are now run through the TCP MLP NN. These TCP test datasets are modified versions of the original datasets and contain only TCP packets. For example, a general dataset starting with 10,000 packets is left with 9000 TCP packets after processing. The 132-150 input features are retained but the hidden and output layers are changed to 23 based on the number of classes. Learning rates of .0001-.000005 and batch sizes of 128-512 are used in experimentation. The classes are completely different as they are based on TCP messages only. Each TCP dataset is tested for accuracy and then run through a scan detector set of functions.

Both models are trained on an NVIDIA Quadpro P1000 GPU. Training time for the general classifier is approximately 4 min/1000 iterations. The TCP classifier training time is longer (7min/1000 iterations) as TCP training set is larger containing 102k packets. Once trained, processing of the test datasets takes 3 seconds each which shows one of the major benefits of a trained neural network model compared to rule based approaches.

Beginning with the handshake messaging, the scan detector looks for a several patterns in attempting to determine whether or not a scan is in progress. TCP connections complete the SYN, SYN-ACK, ACK sequence prior to the exchange of data. Scan attempts are difficult to detect as they do not complete the handshake. Another important part of a scan is the behavior of a closed port. When sent a TCP datagram with the SYN flag set, any response offered by a closed port will set the RESET flag. Typically both the RESET (RST) and ACK flags are set. Since valid clients do not connect to a wide variety of ports, several messages with the RST flag set may also indicate a scan. Lastly scanners sometimes send messages probing the target behavior and so unusual flag combinations (ex. SYN, FIN, PSH, URG) may also be seen.

7.2 Classes

The general classifier labels were assigned based on the most common communication protocols layer 2, 3 and 4 encapsulation. For example; 802.3 or Ethernet II, IPv4 or IPv6, ARP or Management, upper layer protocol (TCP or ICMP or UDP, etc.) These classes are established based on observation of contemporary traffic captures.

The TCP classes are established based on flag patterns seen in contemporary datasets and scans created for the work. Assigned labels based on the TCP flag hexadecimal patterns include; SYN, SYN ACK, ACK, FIN ACK, ACK PSH, ACK RST, RST, SYN ECN CWR, SYN FIN PSH URG, FIN PSH URG and ACK PSH FIN. Internal and external traffic patterns both have a set of these labels.

7.3 Datasets

The test datasets have few restrictions though the training datasets must be comprehensive. For example, the general classifier should have a mix of the traffic types likely to be seen on a modern communication network. In addition, some are dedicated TCP port scan datasets. The TCP training dataset is a capture of TCP packets only and these are generated using a web, mail, game traffic and scans. With this approach, neither of the MLP neural networks are at risk of overfitting or excessive misclassification problems.

There are a few other concerns when attempting to detect scan attempts. Since scanners may scan over time to hide the activity, the datasets tested should be sequential or very large. For sequential datasets, establishing connections between the datasets may be desirable. Lastly, experiments have a priori knowledge regarding the addressing used in the target network. It is assumed that the attacker is from the outside (external to the router) and so we distinguish between inside and outside addressing in the classifiers. This means that within the scan parser, we optimize by eliminating packets that do not fit the attack profile. For example, inside scans are not the focus and so TCP SYN messages from inside sources are ignored. However, this is the only "rule" in use.

7.4 Results

In a performance review the following are measured; overall classification accuracy, individual label accuracy, false positives and misclassification. The accuracy percentages for the datasets are shown in Table 7.1. These values result from the general classifier running for 1500 iterations with a learning rate of .000005. The training set size was 31,642 packets. Dataset 30 values are a simple check since it is the trainer. Once the general classification has run and the TCP packets have been selected from the above named datasets, the TCP classifier runs. The results of the TCP classifier are also summarized in Table 7.1. The TCP Accuracy values are from a trial with a learning rate of .0001 and 1000 iterations.

Table 7.1: Dataset Accuracies

Dataset	General Accuracy	TCP Accuracy
30	99.99	99.31
31	99.99	97.9
32	99.92	99.8
33	98.81	95.55
34	99.69	95.67
35	99.9	96.35
36	99.79	99.67
37	99.98	99.94

As can be seen the accuracy is very good with the lowest at 98.8%. This helps ensure that the next stage will not be hindered by early misclassification.

In addition to overall accuracy, it is important to test the individual class accuracy. Examples from Dataset 34 are shown in Table 7.2. As can be seen, the per class accuracy is high. The Wireshark packet values are directly from the capture program itself and the model packet numbers are the predictions from the neural network. Performance for the other datasets is very similar. Table 7.3 depicts a similar check with the TCP classes (based on flag combinations) from an individual dataset. For space, only some of the categories are shown. In addition, some categories are summarized because some flags exist in more than one class. For example, the SYN flag appears in both SYN,

Table 7.2: Dataset 34 Class Accuracy

Class	Wireshark	Prediction
ARP	363	363
IPv4	6611	6611
ICMP	2	2
IGMP	289	289
UDP	1753	1776
TCP	4543	4544
IPv6	472	476
LLC	2198	2198

SYN-ACK and scan specific messages.

Table 7.3: TCP Class Accuracy

Label	Wireshark	Model
SYN	5286	5287
SYN ACK	50	50
ACK	18292	18098
FIN ACK	17	17
RST	5042	5044
SYN ECN CWR	2	2
SYN FIN PSH URG	2	2
FIN PSH URG	2	2

Lastly the model is run on several scan datasets as shown in Table 7.4. The examples (Datasets 36 and 37) come from a pair of scans generated from NMAP, one targeting a small number of ports and the other 5000.

In the first scan, NMAP performs a stealth scan and OS detection against the target along with a test of 1000 ports. NMAP sends 1112 packets and receives 1113 from the target. Over the course of the scan, a few unrelated TCP packets are created by the target. NMAP discovers that ports 135, 139 and 445 are open. Given this information, it is expected that a capture file would have at least 2225 packets (2413 actual) and that half would be TCP SYNs and the other would be a combination of SYN-ACK and RESET-ACK

Table 7.4: Scan Results

Label	Wireshark	Model
Dataset 36 SYN	1091	1087
Dataset 36 SYN ACK	77	77
Dataset 36 RST ACK	1041	1044
Dataset 37 SYN	5234	5233
Dataset 37 SYN ACK	50	50
Dataset 37 RST ACK	5029	5030

messages. Similar analysis can be applied to Dataset 37 in which the scan sent 5239 packets and received 5049 with an actual total of 10486. The scan was against the same target and so the same ports are open.

The success rate from the scan detector is extremely high. In both cases, the correct open ports were identified from the classification effort. The prediction accuracy for these two scan datasets exceeds 99% with dataset 37 actually achieving 99.9%. This also allows the detection of scans that are not common such as those setting the FIN-SYN-PSH-URG or the SYN-ECN-CWR-RES flags at the same time. In an entire scan there were typically only two of these unusual packets captured. The resulting numbers indicate few misclassification errors and even fewer false positives. The identification of completed handshakes has similar accuracy though is not as important to scan detection.

7.5 Discussion

This two stage classifier is extremely effective at predicting a wide variety of labels even though the TCP labels are much more challenging. To arrive at a model that achieves low loss rates, a wide variety of tests that varied NN structure, hyper-parameters and output classes were completed. Output class labels selection is challenging because they can be based on packet headers, packets carrying data, TCP options and source IP addresses. In addition, packet behavior or characteristics can change for a particular class. For example, when the ACK flag is set, the packet may or may not have data which varies the length field value. The same can be said of TCP options. In addition, the variation between packets is what allows for the classification, but when isolating all of the TCP traffic, this variation is effectively reduced.

In the end, simpler was better. Tables 7.3 and 7.4 demonstrate that the technique of splitting the classification effort and the choices made for labels are very effective at detecting the TCP packets and differentiating between TCP message types. Sorting through all of the flag combinations can be demanding as several of the packets might be counted in more than one place. For example, when assembling those having the ACK flags set (needed to determine is a complete handshake was finished) these packets may be placed in the wrong bin because of the SYN-ACK or RESET-ACK combinations. The scan detector part of the model handles the simple task of processing a collection of lists created by the classifier.

Snort is arguably the most common tool used today to provide a function similar to that of this neural network-based technique. A powerful and flexible tool, Snort has a built in port scanning module (sfPortscan) which works with the Stream filter and scan.rules file. While Snort is fast and accurate, it suffers from the problems of rule based systems. Most notably a lack of adaptability and that human expertise to create complex rules is required. Programs like Snort cannot learn from the environment, can be complex to configure and require dozens or even hundreds of *correct* rules. The Snort documentation also states that increased "sense level" may result in more false positives. A neural network does not use rules, only input and output. Decisions are made based on a probability calculation from a function like Softmax. Once trained, neural networks are as fast or faster than rule based systems. They can also respond to dynamic conditions by retraining without reconfiguration.

7.6 Chapter Conclusion and Contributions

Neural networks have proven to be good function approximators in a variety of fields. Recently they have been deployed successfully to classify network traffic. In this paper we have shown that sequential neural networks can learn the environment and then break up complex tasks. This separation facilitates finer control for protocol classification and achieves accuracy rates above 99%. This is true for overall dataset classification as well as individual classes. We have shown that they can be highly effective in TCP classification and detecting TCP port scan attacks with a level of prediction accuracy matching the general classifier. A review of the results reveals that the incidence of false positives or misclassification is also very low. This success also means that neural networks can be deployed to aid in a variety of security challenges facing communications today.

This was also the first attempt at combining neural networks so that they might work together in a particular application. The success of this particular project led to some of the successes achieved in later investigations. Most notably, the ensemble used in the beginning stags of the Mean Opinion Score Predictor was built with this project in mind.

Chapter 8

CNNs and Network Visibility

8.1 Packet Image CNN Processing

The structure and operation of Convolutional Neural Networks (CNNs) is more fully described in Chapter 4. This chapter discusses the use of CNNs for packet classification and their potential for use in more complex communication applications. This is a departure from the experimentation with MLPs which formed the basis of the previous work.

The preliminary data parsing process is similar for CNNs and is typically completed in approximately 3 sec depending on the dataset sizes. However, since the CNN acts on input different data (typically images), the preprocessing stage does add the step of converting packets to images which adds a few seconds. Recall that incoming packets vary in size and purpose. In a network comprised of an Ethernet substrate, the frames are limited to payload sizes between 46 and 1500 bytes. The IP datagrams are encapsulated in this frame. The neural network selects a frame size and truncates all the incoming traffic data based on this value.

The work described in Chapter 5 achieved exceptional results with packet sizes ranging from 125-200 hexadecimal features. This corresponds to the first 64.5-100 bytes of the packet. For this CNN portion of the work, 196 features were chosen as a size that encompasses a majority of the significant header fields and can be used to create a square image whose first line is the entire Ethernet header visible after capture. With the exception of IPv6, the fixed selection of 98 bytes encompasses all of the data necessary to classify traffic.

Though the dataset packets are still organized into a matrix. At the time of the investigation, the training dataset had 70000 packets. The resulting matrix was [70000,196]) and processed as described previously. The packets are converted into 14x14 images and the matrix restructured to [70000,1,14,14]. The square images make the preprocessing straight-forward and the 14x14 size performed better than 15x15, 13x13 12x12 images. An example of a single Ethernet encapsulated ARP request packet image can be seen in Figure 8.1.

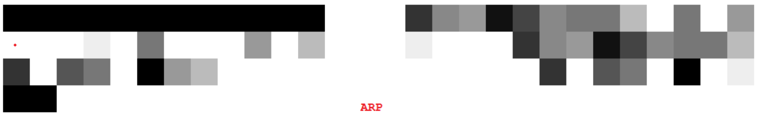


Figure 8.1: ARP packet image

This image is organized along the size of the Ethernet header with each row equal to 14 bytes (28 features). Note that the preamble and frame check sequence are not seen by packet analyzers. This places the entire Ethernet II header on the first line. The gray scale shades are based on the hexadecimal value. ARP requests use broadcast addressing and in an image such as this, it becomes clear that the destination MAC address is FF:FF:FF:FF:FF:FF. The next six bytes are the source MAC address which is followed by the layer two type or 802.3 length. This image also provides some insight into how a CNN might be able to distinguish between packet types as the images are so distinctive. In the case where the packet is smaller than 98 bytes (196 features) it is padded with 0's.

Figure 8.2 depicts an image of a larger packet which comes closer to filling the 14x14 matrix. The start of the IP and TCP headers are indicated. The white space below the packet is the additional padding used for packets $<$ 98 bytes in size.

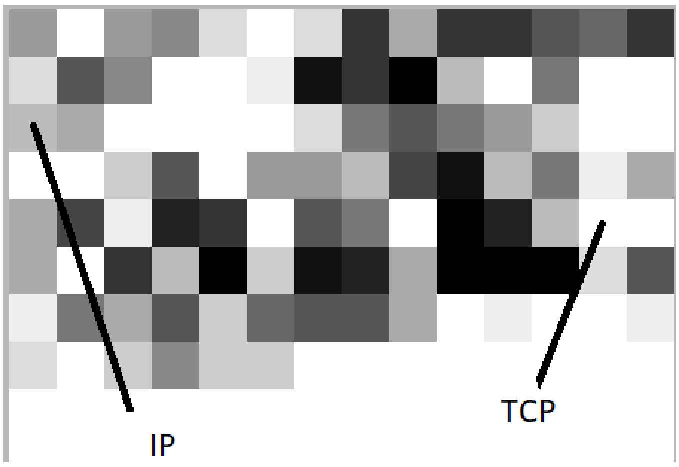


Figure 8.2: Full packet

8.1.1 Larger Images

While the [70000,1,14,14] matrix can be used to accurately classify traffic, another contribution of this work was to address network visibility using CNNs. Thus the preprocessor take an additional step with the data and organizes it into larger images measuring 140x140 hexadecimal characters by taking 100 of the 14x14 images and rearranges them. This is done to provide a temporal snapshot of network traffic over a slice of time. The rationale for this is explained later in this chapter.

8.1.2 CNN Operation

During the CNN forward pass, the packet matrix is run through a series of convolutional and fully-connected layers. One aspect of a convolutional neural network is that it reduces the number of features between layers. For example, a pair of fully connected (FC) layers is a linear combination of all nodes in one layer to all of the nodes in the next layer. The convolutions and max-pooling used in CNNs significantly reduce this number.

CNNs accomplish this through the use of filters which pass over the image and apply a transform for each step of the pass. The transform result is a linear combination. For example, the first convolutional stage of our network uses a 3x3 filter and a step of one. The 3x3 filter (f) starts in the upper left hand corner of the packet image covers a portion of the 14x14 image. The filter then steps to the right and eventually returns to the left moving down as needed to eventually pass over the entire original image making calculations as it goes. The size of the step is referred to as the stride (s). The result is a new 12x12 image. No padding (p) is used for the images. The resultant size can be determined by the formula:

$$((n - f + p)/s + 1)$$

For these images: ((14-3+0)/1 + 1) = 12. Following the convolutional layer (conv1), a max-pooling layer further reduces the number of features. The max-pooling layer performs the same passes, though in this case, the filter is smaller (2x2) and the stride is doubled. The value obtained from each step is the maximum (argmax) for the portion of the image covered by the filter. After the first convolutional and max-pooling layers have made their passes, the resultant image is 6x6: ((12-2+0)/2)+1 = 6.

This process is repeated with a series of additional filters. Each filter does the same thing though they are initialized separately. Filter calculations are combined later in the model. As an example, the first layers of our model use six filters and the subsequent layer uses 16. Another smaller example of a CNN with max-pooling and a series of filters is shown in Figure 8.3. For space, the image starts as a 10x10, is reduced to 8x8 ((10-3)/1+1=8) and then after max-pooling with a 2x2 filter the image is 4x4. There are also six different filters applied to each image.

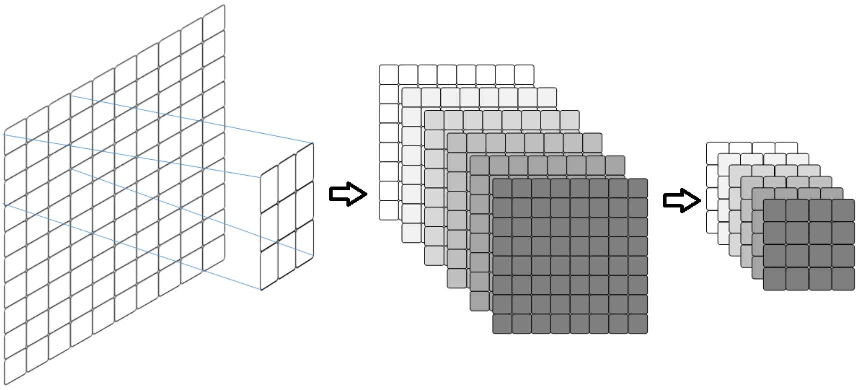


Figure 8.3: Convolution with max-pooling

As the actual 14x14 packet images are small, the first stage is only followed by a one other stage (conv2, max-pooling) which further reduces the images to 2x2. Because of the additional filters, there are a number of these results for each packet. These results are then combined in a series of fully-connected (FC) layers. Fully connected layers rearrange the data once again to a column vector. The column vector size is the product of the image resolution and the number of filters. For this configuration: 2*2*16 filters=64. A series of fully connected layers reduces this value to the number of classes.

An example of this part of the network can be seen in Figure 8.4. Fully connected layers eventually reduce the column vector to a size equal to the number of classes.

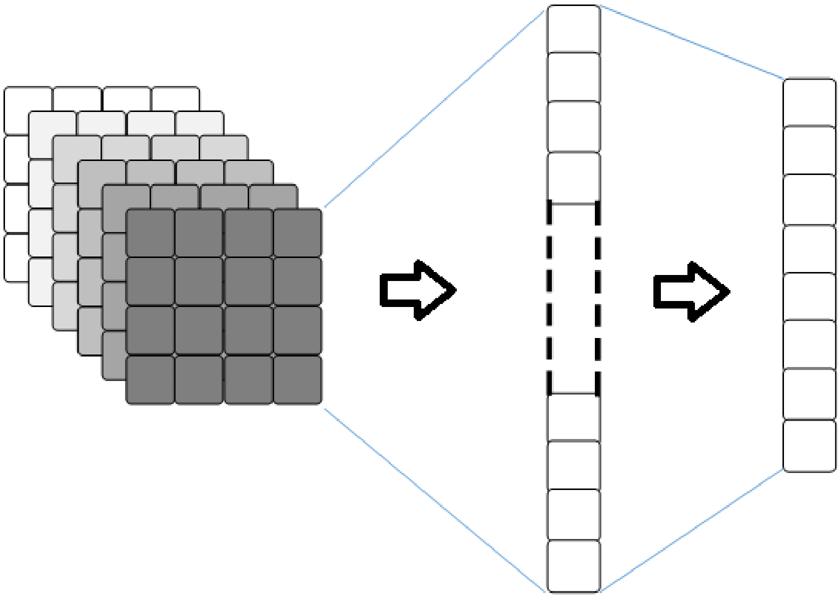


Figure 8.4: Fully connected layers

At the end of the forward pass, a Softmax function then calculates a class probability. Once the error is calculated and back-propagated, the model is run again until either the desired number of iterations has passed or the error rate has declined below an acceptable threshold. The full model can be described as conv1 (3x3x6), max-pooling (2x2, stride=2), conv2 (3x3x16), max-pooling (2x2, stride=2), FC1(64:32), FC2(32:16), FC3(16:14). During training 1000 to

2000 iterations are typically run. While the training is off-line, the architecture itself can then be used on stored datasets or during real-time capture.

8.1.3 Datasets and Classes

As previously stated, the training set contains 70000 packets obtained from an operational network testbed. This is a balanced dataset of fourteen classes, each having 5000 packets. The classes are chosen to include a variety of traffic from layers 2, 3 and 4; TCP port 80/ 8080/443, UDP DNS/DHCP/SSDP/NBNS, CDP, ICMP, IGMP, STP, loopback and ARP. An empty or missing class is also included. A smaller (7000 packets) balanced validation set is also used.

The validation set is followed by four test datasets of varying size and composition. Once the model is trained, the datasets are run through the CNN model. The analysis is run on each section of the larger (140x140) images and the class accuracies and the overall accuracy *for each section* are recorded. A packet map is also created so that packet location assignments and timestamps can be retrieved and reported.

8.1.4 Optimization and Features

A variety of optimization techniques (with and without weight decay) were evaluated in this CNN model. While several including Adam, Adamax and Adagrad, performed well, the highest and most stable performer for this particular configuration was Adadelta with weight decay. The feature size of 196 is based on two factors; the number of packet bytes necessary for acceptable classification accuracy and the desire to use a square image for filter traversal. It was also found that images smaller than 14x14 resulted in an accuracy drop. Larger sizes did not improve performance beyond that experienced at 14x14.

8.2 Visual Representation of the Network

There are several benefits to using neural networks for applications such as data network traffic processing. Neural networks are flexible in that they can process a variety of sources and can be modified for new conditions. Once trained they are very fast, out-performing attempts at parsing through the same amount of traffic. In addition, for the classification work, specialized software is not required. The benefit of deploying convolutional neural networks is that the power of video and image processing can now be used to interpret packets and patterns seen in communication topologies. Thus, the next phase of this work was to combine individual packet images into a larger picture that could depict network behavior at a point or points in the topology, providing an avenue to infer network conditions.

The larger image is run through the same model but rather than testing a single image, the CNN is run against each section of the combined image. Since the CNN is already trained, it quickly determines the content of each section of the larger image.

A typical network conversation, performance problem or attack typically occurs within a certain period of time. A larger image would be able to depict the packets and derive the relationship between them because it is a slice of time for the network. For example, the PING command issues ICMP echo requests and waits for ICMP echo replies. If the ARP tables are missing the requisite entries, an ARP exchange is triggered prior to the ICMP messages. An 2x2 grid image with these packets is shown in Figure 8.5.

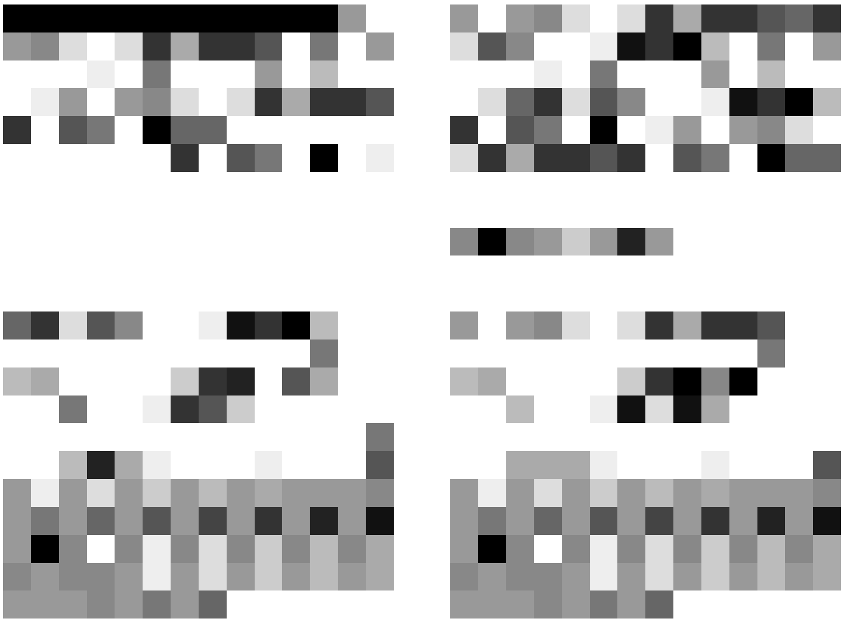


Figure 8.5: ARP and ICMP exchange

A close examination of these packets reveals the differences in the source and destination MAC addresses and in other parts of the message such as the alphabetical content of an ICMP request. Even in this small grid, the entire exchange is captured. This image represents standard behavior and this approach may be able to recognize anomalous patterns. Currently the CNN can recognize packet classes and excessive delay between packets (indicated by a missing packet), all while completely unaware of field meanings, headers or content. A system would thus be able to classify all of the packets and patterns or latency issues could be recognized.

A larger image of a packet series provides a snapshot of network activity over a larger slice of time than a single packet would. The average delay between the packets seen in Figure 8.5 is 3 msec. If packets were processed together in a larger image, the span of time covered by the image would be the number of packets x 3 msec. giving a broader view into the packet flows in both directions. Further, a series of these larger images could be thought of as a series of video frames, each covering that time span and conveying much more information. Even in this small image the information contained is 4x a single packet. In our larger tests, one hundred 14x14 images are combined sequentially into a 10x10 grid (140x140 features). This 10x10 grid now represents a snapshot in time of .3 seconds. This can be repeated at several interfaces on the same network element or at different locations within the topology.

Another example can be found in the identification of potential bottlenecks or sources of latency. Latency (along with jitter and packet loss) is a primary cause of poor application performance. Given the number of network connections that run over wireless networks, upcoming 5G deployments and remote cloud services, this is going to continue to be a significant challenge. To help address this, the CNN is capable of identifying spacing or missing packets. Next steps might include recognizing patterns in this collection.

If the packet stream is coupled to the time stamps, a delay might cause the traffic image to look like the one shown in Figure 8.6. This example actually comes from common router behavior at the beginning of a flow. The first ICMP echo reply was lost because the router was busy populating its ARP tables. In this case, once a latency threshold is exceeded the packet is not drawn in the next time slot. The latency threshold is a configuration parameter of the system.

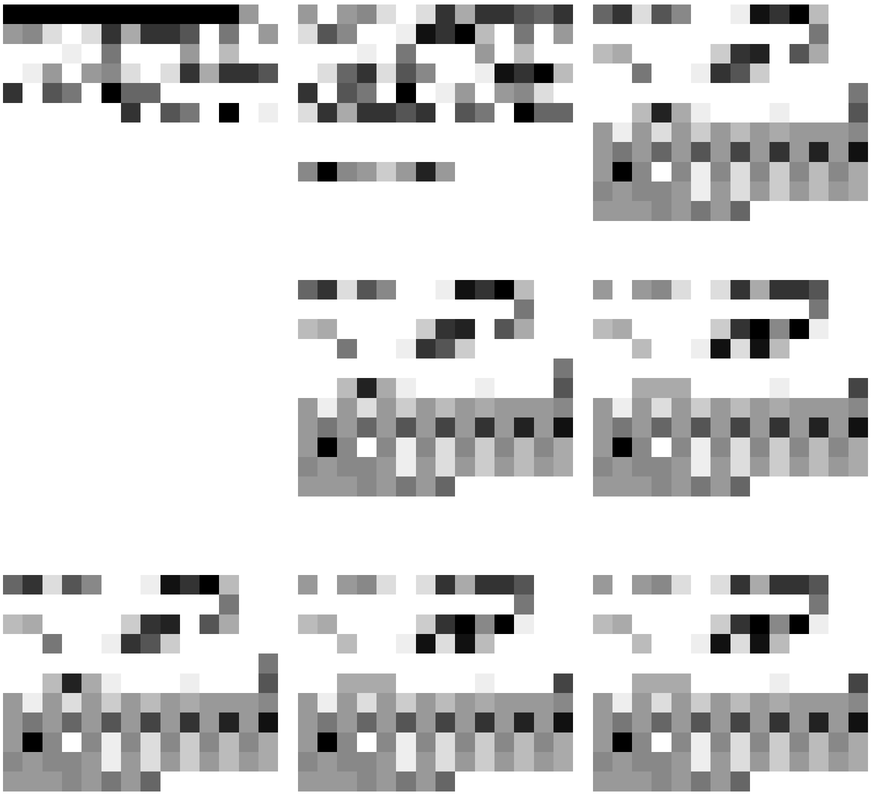


Figure 8.6: Missing packet

The size of the larger image of sequential packets is also a configurable parameter in the model. The larger the image, the greater the potential to expose other patterns to gain visibility into network behavior at a given point. Depending on threshold values, problems with excessive latency or packet loss might present as a collection of white spaces. Excessive jitter or intermittent problems might be more easily diagnosed by observing the surrounding packets and white-space. Since the CNN model can identify both the packet types in the conversation and the latency locations in the traffic images, causality and direction might also be established.

Again, the model used here would be able to provide highly accurate classification of individual packet classes and potentially events. In an incident response scenario, a person reviewing static examples from the stream might be able to spot problems in much the same way graphs are currently used for visualizing latency, capacity and busy hours on the network. The difference is that this technique provides a view that is multi-dimensional in that it shows packets and the relationships between them.

A common approach to address security concerns is to build attack profiles. A similar template could be developed and this identification could result in recommendations for security or quality of service actions to be taken. A collection of these large image streams could provide insight into behavior across an entire set of network interfaces.

8.3 Results

A CNN model has a number of configuration options that include optimization techniques, number/size/stride of filters, learning rate, etcetera and many of the related works do not provide these details. Table 8.1 depicts results from model filter variations. The columns represent the number, size (Sz) and stride (St) of the filters. All of these results are from tests against 14x14 packet images and using the AdaDelta optimizer. Due to stability, the first configuration was chosen for subsequent tests. While these configurations all show results exceeding 95%, it can be seen that the first two achieved 98.7 and 99.45% respectively. The choice of optimizer can be task specific and so a comparison between the several of them was also completed. Optimizer performance varies with configuration. For example, AdaDelta performed well for the filter configurations shown in Table 8.1 however performance dropped in other formulations. The results shown in Table 8.2 were achieved after reducing the

Table 8.1: Filters vs. Accuracy

Conv1			Max P		Conv2			Max P		
No	Sz	St	Sz	St	No	Sz	St	Sz	St	Acc
6	3	1	2	2	16	3	1	2	2	.987
6	3	1	2	2	16	3	1	2	1	.9945
6	3	1	2	2	16	2	1	2	1	.962
6	2	1	2	1	16	2	1	2	2	.951

learning rate to 1e-6 with the filter configuration indicated in line 1 of Table 8.1.

Table 8.2: Optimizer Comparison

Method	Train Time	Iterations	Ave Acc
AdaDelta R	4hr 33min	1000	.993
Adam	1 hr 32min	1000	.916
Adam R	1hr 4min	1000	.893
Adamax	1hr 7min	1000	.751
ASGD	1hr 2min	1000	.939
AdaGrad	59min	1000	.864

The "R" after an optimizer indicates that it used regularization or weight decay. In the end, consistency in accuracy were the determining factors in choosing an optimizer. It should be noted that changes to classes or structure can require a review of all parameters. Additional tests were run for various learning rates (1e-3, 1e-5) and for longer periods of time but stopping at 2000 iterations. Again, due to stability, the first configuration shown for AdaDelta in Table 8.1 was chosen. The model also proved its effectiveness when the results of the various classes returned. For space, Table 8.3 provides a sample of the values for test Datasets 3 and 4. The paired columns depict the actual packets seen in a Wireshark capture vs. the number of packets predicted by the model for each class. For example, Dataset 3 had 447 ARP packets compared to a prediction of 439. Dataset 4 had 200 observed packets compared to 199 predicted. The overall accuracy for these datasets were 99.7 and 99.1% respectively. The average for all datasets was 99.2% as the other datasets had

Table 8.3: Class Accuracy

Class	Wireshark 3	Pred 3	Wireshark 4	Pred 4
ARP	447	439	200	199
IGMP	8	6	199	199
STP	56	56	1050	1050
DNS	268	290	738	732
SSDP	15	15	330	332
TCP 80	618	594	11931	12163
TCP 443	24433	24415	12463	12208

similar results.

8.4 Discussion

The contribution of this CNN approach to challenges associated with modern communication networks is not necessarily the structure of the CNN but rather in the application of CNN to the problem of accurate individual packet classification. Researchers have used various machine learning models to address specific areas but few have tackled this core need directly. Even fewer research projects have gone to this level of accuracy. The architecture and data organization used here has proven successful in this task and when CNNs are used, the training time is often reduced.

Over the course of testing occasional aberrations occur with the packets. For example, errors in the classification of spanning tree led to the discovery that there were two different versions within the datasets. ARP was also misclassified as a management protocol due to the inherent padding used in the messages. These were addressed by clarifying class definitions using operational codes or balancing the datasets.

Once the CNN is established as the successful central component, larger problems such as latency detection or mean opinion scores associated with real-time traffic such as VoIP can be addressed. By re-imagining packets and packet flows as images as shown in Figure 8.6 contemporary image and video processing techniques based on the CNN structure can be applied. The larger images provide an temporal slice of network time offering a perspective for viewing network traffic or missing elements. An approach like this can aid in visualizing network conditions making easier to understand the current scenario or events during network transients.

8.5 Chapter Conclusion and Contributions

Many of the challenges addressed in network research today have at their core a need to classify communication packets. The central contribution of this work a CNN architecture that can be used to not only classify traffic with a high degree of accuracy but aid in improving network visibility to address application performance problems through visualization has been presented.

Attempting to utilize CNNs by converting packets to images is not trivial because the information format (and they way we think of the data) is fundamentally different. It was by no means a foregone conclusion that the approach would work and several formatting attempts were made before arriving at the current configuration. Once this was successful, by treating packets as images, contemporary image processing techniques can be applied. These individual packets can also be combined into larger images that can provide insight into a activity during a time span. This increased visibility can help detect performance problems and potential security threats.

Using this model, individual class identification accuracy exceeds 99% and images combined into a grid to improve visibility can be identified with the same high accuracy levels.

Chapter 9

Ensembles and the Mean Opinion Score

9.1 Approximator to non-Approximator

Many of the challenges addressed by communication network research rely on the ability to identify packets and determine relationships between them. This is also true for the operation of many devices such as firewalls. Even the standard processes such as routing require that the traffic is processed to some level prior to handling. Additional processing must be accomplished if other decisions are to be made. For example, routers can be used to enforce a variety of policies that examine packet header fields. These fields can only accessed through some form of parsing or table lookup. With the advent of greater compute power and advances in machine learning, it may be possible to address a wide variety of challenges with what was previously called an approximator. With the appropriate supporting architecture it may even be possible to achieve near perfect recognition rates so that downstream, highly effective applications can be developed.

The near perfect accuracy is a requirement because the subsequent decisions depend on the answers from earlier stages. For example, a variety of security challenges such as port scanning, data ex-filtration and anomalous traffic still plague networks today. A packet permitted or denied based on a bad decision made by an algorithm could lead to either malicious traffic entering the system or valid traffic being dropped. In the case of quality of service, traffic might be unnecessarily delayed or elevated above higher priority traffic due to a misclassification error. Traditional approaches are not immune to errors when used for these applications. Misconfiguration or incomplete rule sets can create additional problems as well. In this work, ensembles are used to achieve high accuracy. Toward the end of the chapter the targeted application is a Mean Opinion Score (MOS) predictor.

9.2 Accuracy

Deterministic structures such as decision trees or table lookup can be used but often at the cost of speed. In addition, these may require specialized software. However, a trained machine learning model such as a neural network may be able to achieve the level of accuracy desired and at a speed that does not degrade performance or cause an excessive lag in a monitoring system. In Chapter 5 a multi-layer perceptron neural network was used to successfully classify packets with an accuracy that exceeded 99%. Further, several of the experimental MLP models were found to have this high accuracy rate.

Chapter 8 presents convolutional neural networks that were deployed for the same task. Using CNNs to classify packets results uses fewer parameters during the forward pass which can make the memory use and training more efficient. The CNNs were also able to exceed 99% accuracy. Like the MLP models, several CNNs reached this high level of performance. Through experimentation with the optimizers, tuning hyper-parameters and trials with model depth and width, an MLP or CNN model will occasionally achieve 100%, though not consistently.

Both the MLP and CNN models were then inserted into application system that addressed a particular challenge. The MLP networks were deployed in a sequential architecture that, when combined with a parser could detect port scans by examining TCP flags. The CNN model was part of a system that examined a collection of packets that represented a particular temporal space. The CNN might then be used to recognize patterns in network behavior or problem areas using techniques borrowed from image processing.

Based on these successes, the following base architecture is proposed: an ensemble of two CNN and two MLP models that together will determine the correct classification. Each model makes a determination as to the correct label for each packet. The results of the ensemble are compared and a majority vote selects the final class label. Should any model reach 100% without rounding, that model is chosen as a class selector. Testing reveals that after the tuning, it is common for at least one model to reach 100% accuracy.

9.3 Classes

Part of the reason for the success is that the neural network stages are deployed sequentially. The base structure is performing general classification in that it handles broad categories such as TCP, UDP, ICMP, etcetera. Once this is accomplished, the architecture moves on to the specifics of each category. For example, the UDP classifier moves on to DNS, DHCP, SSDP, etc.

Another aspect of the tuning for the model is that observations are made on actual network traffic in order to determine the types commonly seen. For example, there are many ICMP messages defined by RFC 792 yet only a subset of these are actually seen on a contemporary network.

9.4 Datasets and Dataset Recursion

Current datasets have been either captured or constructed of traffic seen on the local test-bed. The available traffic has been divided into training, validation and test sets for each of the general categories. Much work has gone into ensuring that the datasets are also balanced. That is, having the same number of samples per class. The various models can be pre-trained or trained at run-time using the combined training dataset which currently contains 85000 samples. These samples are also used to train the TCP and UDP models. A complete discussion on the approach used with the data and datasets can be found in Chapter 3.

Once the base classification is made, the packet numbers are recovered and the test datasets are restructured into general category datasets. This emphasizes the need for highly accurate classification early in the process. Thus, a general dataset made up of a variety of traffic types would then be split into smaller TCP, UDP, ICMP, etc. datasets. Subsequently these would be run against the associated trained models. Each time this occurs, the sample with its packet number and timestamp are paired.

9.5 Feature Selection

This neural network ensemble processes raw packets. The models take in complete packet headers (Ethernet, IP, ICMP, TCP, UDP, etc.) plus some portion of the packet payload depending on the number of features desired. For this work, 196 features or hexadecimal characters (98 bytes) are used in the CNNs. The MLPs vary in the number of features used as input. This number is used to satisfy the operational network structure and accuracy goals. Packets smaller than the desired features are padded. The timestamps and packet numbers are also stored for use in particular applications.

9.6 Operation

Other than the restructuring of datasets based on class, the operation of the NN models is exactly as described in Chapter 4. The first stage of the system architecture is depicted in Figure 9.1.

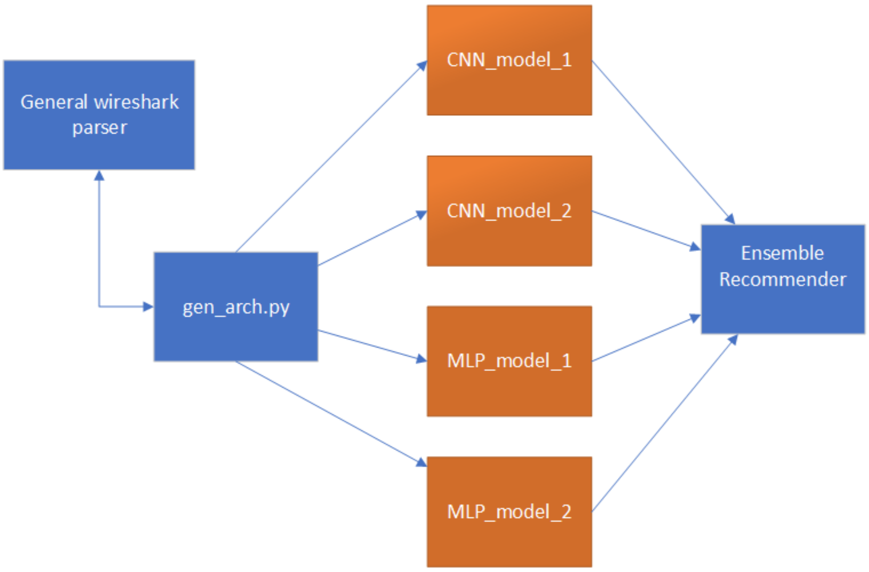


Figure 9.1: Ensemble First Stage

Packets are preprocessed in the general wireshark parser and then each of the models is trained using their own tuned parameters. Currently the training occurs sequentially but this can be done in parallel using multiple GPUs. After each is trained, the results are stored and then these results are reviewed by the recommender function. As mentioned earlier, should a model score 100% accuracy, it is immediately used for general training. If this is not the case, majority voting is completed on each packet for the final determination. Because of the reduced number of classes (10) now used in general classification, majority voting is not usually necessary.

The structure of each MLP neural network is similar though they vary hidden nodes and batch size. CNNs operate in a slightly different fashion and so their current difference is simply the optimizer in use with CNN model 1 using Adam and CNN model 2 using Adamax.

Once the recommender function is complete, the general classes of packet can now be run against the specific types of packet for that class. The TCP classifier will further divide the packets into port 80, 8080 and 443 traffic. In addition, the UDP classifier will label (DNS, DHCP, etc.) the associated classes. The architecture also attempts to reuse much of the code and so each of these sections runs the particular datasets through the same set of neural networks. This portion of the architecture is shown in Figure 9.2.

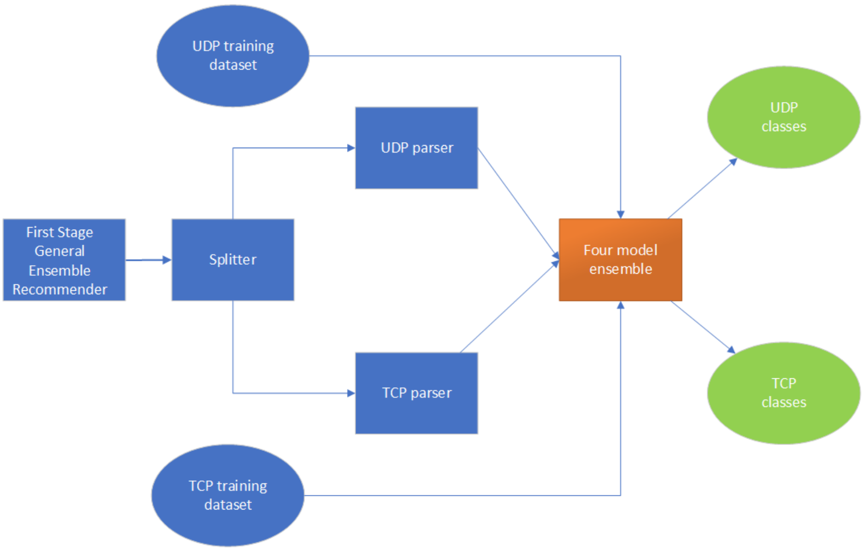


Figure 9.2: Ensemble First Stage

As can be seen, the architecture is modular and can be expanded to include the other general categories of traffic.

9.7 Applications

Neural networks have been shown to be accurate classifiers of communication traffic. In addition, they are fast when trained and can take a variety of data sources as input without the need for additional software. Another advantage of the system as constructed is that it does not require controlling either of the end points - the traffic is from a bi-directional capture source. However, packet classification, while a valuable contribution, is not the only goal of this work. A important facet was to determine the applications of the architecture beyond accurate labeling of individual packets.

In Chapter 7 a similar idea was explored to help combat a real and preset threat for all communication networks. In Chapter 8, packets are converted to images with the goal of not only improving performance of the system but identifying patterns within the data stream. In the next section another application will be explored: Mean Opinion Score prediction for Real Time Transport Protocol (RTP) traffic used in voice connections. This is subclass within UDP and the system has been expanded to address Voice over IP behavior.

9.7.1 Voice over IP

Voice over IP (VoIP) is a critical part of the infrastructure for many organizations. Voice over IP packetizes voice traffic into Real Time Transport Protocol (RTP) streams. A signaling protocol such as the Session Initiation Protocol (SIP) determines the connection properties and the port numbers to be used by RTP.

Close attention is paid to the performance of the VoIP system. Most notably the latency, packet loss and jitter are measured as these directly impact user experience. User experience is measured using the Mean Opinion Score or MOS.

Within the proposed system, RTP is one of the classes identified. Since RTP is encapsulated in UDP, when the UDP packets are removed from the general test datasets, RTP further split off for processing. The RTP timestamps can be used to describe the relationships between packets and provide a measure of performance.

9.8 Mean Opinion Score Predictor

Mean Opinion Score (MOS) is a quality evaluation given to voice circuits. The values range from 1-5 with higher numbers indicating better quality. As the name suggests, MOS is often a reflection of perceived call quality from the user perspective. With the advent of Voice over IP (VoIP), voice is packetized and these each packets have their own latency and jitter values. These values can be used to provide a calculated MOS rather than rely on feedback from the user. This MOS can then be used to inform the network operator of possible problem areas so that manual or automatic action can be taken.

The traditional telephony E-model calculation for MOS can be found in [8] and is actually derived from the calculation of the transmission rating factor R. This calculation is shown here.

```
EffectiveLatency = ( AverageLatency + Jitter * 2 + 10 )

if EffectiveLatency < 160 then:
    R = 93.2 - (EffectiveLatency / 40)
else
    R = 93.2 - (EffectiveLatency - 120) / 10
R = R - (PacketLoss * 2.5)

if R <= 0:
    MOS=1
if R > 0 and R <= 100:
    MOS = 1 + (0.035) * R + (.000007) * R * (R-60) * (100-R)
if R > 100:
    MOS=4.5
```

Since Ethernet transmission does not have as many of the traditional impairments seen on telephone lines (example: poor signal to noise ratio), the calculation of R is typically based on latency, packet loss and jitter values. Average latency and jitter are based an understanding of arrival time and the associated delays. This means that the calculation must have knowledge of the source and destination transmission times.

The determination of packet loss is typically based on TCP timers at one of the endpoints. Should TCP segments arrive out of order or delayed beyond the timer threshold, the packet is assumed lost. Latency is a measure of delay but again, typically measured at an endpoint. Jitter is variation in expected arrival times and can result in unpredictable performance. Jitter is typically calculated based on the formulas presented in RFC 3550 RTP: A Transport Protocol for Real-Time Applications [60] which takes into account the differences in packet arrival time and the RTP timestamps:

$$\text{jitter} += (1./16.) * (\text{d - jitter})$$

Where **d** is the combined time differences and the new jitter value is based in part on the previous jitter value. Typical target latency, packet loss and jitter values for a VoIP system are 150ms one way, 1% and 5ms respectively.

In this system, the determination of latency, packet loss and jitter values is challenging because it does not have access to the endpoints. Nor does the architecture inject measurement packets. Recall that the packet datasets are captures of bi-directional traffic flowing past a particular point. The goal at this point was to determine whether MOS might be predicted even with these limitations. Thus, the calculations are modified. Once the RTP packets have been separated from the UDP class, the time deltas between packets in the same stream can be calculated in order to get an understanding transmission performance and delay. In the same way a simple moving average of the delays can be calculated.

The ground truth is established using a jitter calculation based on RFC 3550 and an evaluation of the inter-packet delays for a particular RTP data stream. Currently packet loss is not included although increased packet delays that may result from packet loss are.

Since MOS is an outcome or calculation, it can be thought of as another objective function to be approximated by a neural network. Thus, a new model can be created that is a predictor of MOS. A training set of packet traffic with varying levels of delay was created. Once an MOS model is trained, the RTP streams can be run through the MOS predictor. This portion of the architecture is shown in Figure 9.3.

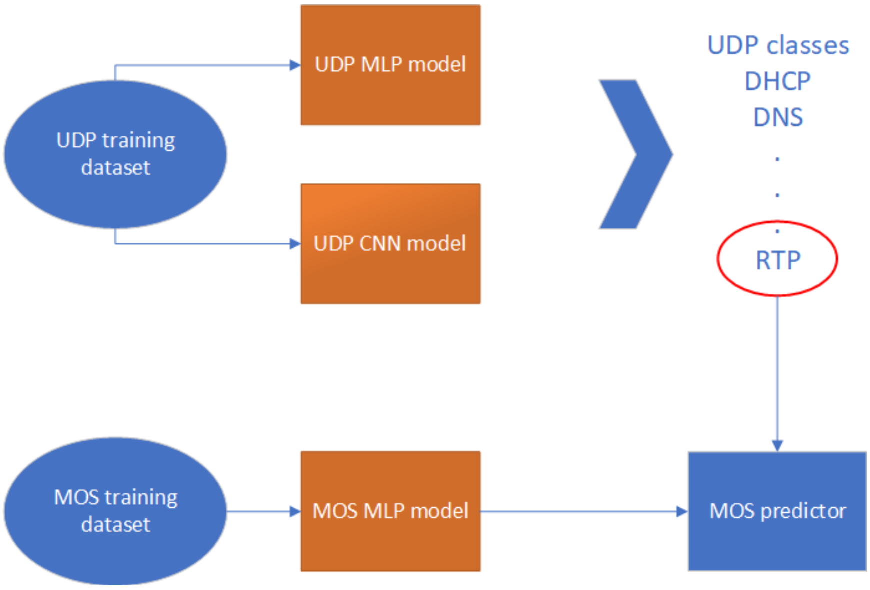


Figure 9.3: MOS trainer and predictor

The MOS model is a MLP neural network that takes as input the packet transmission delta times, jitter and a simple moving average. The output of the MOS MLP model are 4 possible MOS values ranging from 1.0 - 4.0.

For accuracy, the output from the MOS calculation is compared with the result from the model. In this way, the MOS behavior of voice traffic can be predicted from packet streams heading in either direction. This result might be input into a software switch or router in order to modify a throughput or queuing configuration.

9.9 Results

As discussed earlier in this chapter, the stages of the architecture are sequential with the general classification running first. It is critical to downstream stages that this first step is accurate. The recommender compares the results from the four models and either selects a model with 100% accuracy or votes on each sample based on the classification from the models. Typically one or more of the models will have a perfect result. An example of the general classifier output after 1000 iterations is shown in Table 9.1. Once this is complete, the datasets are run through the trained model starting with the validation set. During the run shown in Table 9.1, the validation set accuracy was .99689.

Table 9.1: General Classifier Stage

Model	Accuracy	Count	Correct
CNN 1	.99934	90000	89941
CNN 2	.99953	90000	89958
MLP 1	1.0	90000	90000
MLP 2	.99994	90000	89995
Average	.99971	90000	89973.5

Currently there are eight datasets used int the evaluation of this particular model. Dataset 0 is the trainer, 1 is the validation set and the rest are test datasets. Datasets 7 and 8 are comprised of RTP traffic only. As an example of the MOS model operation, the processing of these datasets will be followed through the architecture stages. Table 9.2 lists the accuracy for each dataset after the general recommender has run. An examination of the class breakdown also aids in following through to the next stage. A breakdown of the initial classes along with the behavior of several datasets is shown in Table 3.2. This set of examples includes the training set (0), general test sets (4 & 5) and

Table 9.2: Dataset Accuracies

Datset	Accuracy	Count	Correct
0	1.0	90000	90000
1	.99689	9000	8972
2	.99426	100000	99426
3	.99246	25847	25652
4	.99602	28145	28033
5	.98666	24354	24029
6	.98995	18610	18423
7	1.0	3349	3349
8	1.0	9459	9459

datasets 7 & 8 are UDP/RTP only.

Table 9.3: Datasets and Classes

Class	Dataset 0	Dataset 4	Dataset5	Dataset 7	Dataset 8
ARP	5000	200	0	0	0
ICMP Echo Req	5000	15	0	0	0
ICMP Echo Reply	5000	15	0	0	0
Loopback	5000	0	2907	0	0
Spanning Tree	5000	1050	14534	0	0
Cisco Discovery	5000	32	484	0	0
IGMP	5000	199	2356	0	0
TCP	15000	25542	0	0	0
UDP	40000	1091	0	3349	9459
Total	90000	28145	24354	3349	9459

The important objectives at this point are the separation of the UDP packets, successful identification of the RTP packets from the new UDP datasets, separation of the RTP packets into individual streams and finally the MOS analysis and prediction. The UDP MLP models are now trained on a UDP specific dataset pulled from the original general trainer. The UDP accuracies for the same 1000 iterations are shown in Table 9.4.

As can be seen, the behavior of the datasets can vary. Newer datasets were

Table 9.4: Dataset UDP Accuracies

Dataset	Accuracy	Count	After first stage	Correct
0	1.0	40000	40000	40000
1	.98717	4000	3974	3923
2	.83096	3169	2597	2158
3	.88075	283	478	421
4	.9002	1091	1012	911
5	.99653	4073	3748	3735
6	.94606	1522	1483	1403
7	1.0	3349	3349	3349
8	1.0	9459	9459	9459

added without a complete update to the working code. The current work is to apply the combined models to all of the datasets which will improve accuracy and stability of the system. However, for the purposes of this chapter, the high RTP recognition rates are sufficient to demonstrate architecture operation.

As processing continues, the UDP classes are identified. This includes the RTP streams. An example of the output is shown in Table 9.5. The RTP classes identify the voice traffic streams. While there is some variation in UDP class accuracy, the RTP streams are readily recognized. The MOS model is trained on another training set build from simple timestamps that vary over a range to provide the different classes. This introduces variation in the timestamps and time deltas for traffic in the dataset.

Table 9.5: UDP classifier output

Class	Dataset 0	Dataset 4	Dataset5	Dataset 7	Dataset 8
DNS	5000	733	868	0	0
DHCP	5000	3	67	0	0
SSDP	5000	330	3111	0	0
NBNS	5000	26	27	0	0
RTP 1	5000	0	0	0	4741
RTP 2	5000	0	0	0	4718
RTP 3	5000	0	0	1676	0
RTP 4	5000	0	0	1673	0

The ground truth for the MOS model is established via the calculation shown earlier in this chapter. The model is trained on the values used as input to the algorithm. After the test datasets (RTP pulled from UDP) are run, they are compared to the ground truth calculations for accuracy. Table 9.6 depicts the values from the datasets that contained the four RTP classes. This includes the training and validation sets as a check.

These results show that the model can be trained to recognize MOS score from data patterns based primarily on the timestamps from the RTP packets. The difference in this part of the architecture is that there is a single MLP model that is trained. This is because an image cannot be created from the low umber of features used for MOS prediction which eliminates the use of CNNs.

Table 9.6: MOS Accuracy

Class	Dataset 0	Dataset 1	Dataset 7	Dataset 8
RTP 1	.998	.98	-	.9979
RTP 2	.998	.98	-	.9978
RTP 3	.998	.98	.994	-
RTP 4	.998	.98	.994	-

9.10 Discussion

An important point through this work is that access to contemporary data that includes raw packets is important to the success of these models. However, the MOS predictor is one of the few places where our current datasets are limited. This is true for the number of classes and streams having a variety of MOS profiles. Currently, only one of the RTP streams has a lower MOS score which raises the concern of over-fitting and an inability to generalize well.

Thus there are three facets which will improve this section further and are part of the future work plan: access to a greater amount of data, further tuning of the models and the application of the ensemble to all datasets.

9.11 Conclusion

This chapter had two main objectives; to create an ensemble that might be used to increase accuracy and to utilize that increase in accuracy for a particular application. To this end an ensemble of two MLP NNs and two CNNs was built that used majority voting in order to determine the proper packet classification. This resulted in an increase in accuracy with some datasets reaching 100%.

Since packetized voice is a part of most major networks, and quality of service paramount, the application targeted was an MOS predictor. This required high recognition rates in the early architecture stages so that the RTP streams could be identified from the UDP traffic. Additionally, the timestamps were retained so that a ground truth could be calculated for each stream. The trained MOS model utilized the timestamps as features and subsequently was able to successfully predict the MOS behavior of the captured streams.

Chapter 10

Conclusion

The projects, methodologies and experiments described in this thesis have focused on the application of neural networks to contemporary communication problems. Previously the lack of processing power and memory made this combination difficult. Upgraded GPUs allowed us to deploy neural networks in both off-line and real time modes.

At the center of each communication traffic challenge is the need to accurately process and classify network packets. The first paper resulting from this work [29] demonstrated that multi-layer perceptron neural networks (MLP NN) could successfully classify (above 99% accuracy) a variety of contemporary traffic types. This paper also sought to provide a structure and methodology that could be used by other researchers. Greater detail regarding this work and general neural networks can be found in Chapters 4 and 5.

Utilizing neural networks in this way is not without its own challenges. Inconsistent behavior and misclassification are regularly appearing issues and there is the constant goal of reducing training time while increasing accuracy. It is common to hear the design of neural networks being called an "art form" because little is published regarding the principles behind their construction. These ideas led to the paper [28] which examined a variety of optimization techniques that might be used to improve performance. An investigation into wider and deeper networks was also completed which showed extremely effective architectures that might be utilized.

Another extremely important idea was also pursued: that of balanced datasets.

It is not usual to see researchers seeking more data over good data. Variations in our accuracy led to the discovery that unbalanced datasets were the culprit. It is far more important to ensure that the datasets are properly constructed and curated. Each class must have the same number of samples in the training and validation sets in order to achieve greater recognition rates and consistency. The Chapter 6 provides the full explanation and experimental results.

Successful packet traffic classification is a very important goal but a fair assessment would wonder about actual applications. Security problems plaque communication networks and so the MLPs were applied to port scans. In [30] the methodology used in addressing port scans is fully described but in this work the neural networks are deployed sequentially. In the first stage, a general classification is completed and this is followed by TCP segment identification. This second stage has a TCP specific training dataset and is followed by a port scan detector. Like the previous work, this structure was able to successfully detect a variety of port scans with an accuracy above 99%. Chapter 7 describes the process and operation of these sequential neural networks.

Neural networks are beginning to receive greater attention but most of the contemporary work utilizes variations in Convolutional Neural Networks or CNNs. Thus, a novel approach to classifying packets was considered: converting packets to images in order to leverage recent advancements in image processing. Further, the idea of collecting images into a larger mosaic that might reveal network patterns is explored. It was found that like the MLPs, CNNs could achieve that high level of recognition and that more comprehensive patterns might be identified as well. The paper [31] and Chapter 8 discuss and define this novel approach.

Ensembles have a variety of uses for packet traffic. Chapter 9 is an investigation into MLP and CNN ensembles that can further increase accuracy, reduce complexity at various stages of the architecture and tackle another problem of communication networks: quality of service prediction. This chapter borrows heavily from the port scan detector but then expands it to provide a general architecture that can be used for almost any application. This is because each general class of traffic (ICMP, TCP, UDP, etc.) can be given its own ensemble. The chapter follows Real Time Transport Protocol (RTP) traffic from the general classifier, through the UDP specific stage until finally we arrive at the RTP packets themselves. The timestamps are recovered and a highly accurate prediction of the Mean Opinion Score (MOS) is first calculated and the predicted. This approach shows much promise and will form the basis of almost all future work.

Bibliography

[1] Norbert Ádám, Branislav Madoš, Anton Baláž, and Tomáš Pavlik. Artificial neural network based ids. In *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 000159–000164. IEEE, 2017.

[2] Omar Al-Jarrah and Ahmad Arafat. Network intrusion detection system using neural network classification of attack behavior. *Journal of Advances in Information Technology Vol*, 6(1), 2015.

[3] Leopoldo Angrisani, Domenico Capriglione, Luigi Ferrigno, and Gianfranco Miele. Measurement of the ip packet delay variation for a reliable estimation of the mean opinion score in voip services. In *2016 IEEE International Instrumentation and Measurement Technology Conference Proceedings*, pages 1–6. IEEE, 2016.

[4] Tom Auld, Andrew W Moore, and Stephen F Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on neural networks*, 18(1):223–239, 2007.

[5] Hitesh Ballani, Paolo Costa, Christos Gkantsidis, Matthew P Grosvenor, Thomas Karagiannis, Lazaros Koromilas, and Greg O'Shea. Enabling end-host network functions. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 493–507. ACM, 2015.

[6] Andrew R Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14(1):115–133, 1994.

[7] Roberto Battiti and Francesco Masulli. Bfgs optimization for faster and automated supervised learning. In *International neural network conference*, pages 757–760. Springer, 1990.

[8] Jan A Bergstra and CA Middelburg. Itu-t recommendation g. 107: The e-model, a computational model for use in transmission planning. 2003.

[9] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[10] Mateusz Buda, Atsuto Maki, and Maciej A Mazurowski. A systematic study of the class imbalance problem in convolutional neural networks. *Neural Networks*, 106:249–259, 2018.

[11] Sung-Bae Cho and Jin H Kim. Combining multiple neural networks by fuzzy integral for robust classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 25(2):380–384, 1995.

[12] Benoit Claise. Cisco systems netflow services export version 9. 2004.

[13] David D. Clark and Mike Wittie. Caida/ucsd.

[14] Carlos García Cordero, Sascha Hauke, Max Mühlhäuser, and Mathias Fischer. Analyzing flow-based anomaly intrusion detection using replicator neural networks. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 317–324. IEEE, 2016.

[15] Gideon Creech and Jiankun Hu. Generation of a new ids test dataset: Time to retire the kdd collection. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 4487–4492. IEEE, 2013.

[16] Mehiar Dabbagh, Ali J Ghandour, Kassem Fawaz, Wassim El Hajj, and Hazem Hajj. Slow port scanning detection. In *2011 7th International Conference on Information Assurance and Security (IAS)*, pages 228–233. IEEE, 2011.

[17] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[18] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[19] Cristian Estan, Ken Keys, David Moore, and George Varghese. Building a better netflow. *ACM SIGCOMM Computer Communication Review*, 34(4):245–256, 2004.

[20] Kieran Flanagan, Enda Fallon, Paul Jacob, Abir Awad, and Paul Connolly. 2d2n: A dynamic degenerative neural network for classification of images of live network data. In *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pages 1–7. IEEE, 2019.

[21] William A Gardner. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal processing*, 6(2):113–133, 1984.

[22] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[23] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[24] Stephen José Hanson. A stochastic version of the delta rule. *Physica D: Nonlinear Phenomena*, 42(1-3):265–272, 1990.

[25] Md Enamul Haque and Talal M Alkharobi. Adaptive hybrid model for network intrusion detection and comparison among machine learning algorithms. *International Journal of Machine Learning and Computing*, 5(1):17, 2015.

[26] Bruce Hartpence. The rit sdn testbed and geni. RIT Technical Report, 2015.

[27] Bruce Hartpence and Andres Kwasinski. Performance evaluation of networks with physical and virtual links. In *Global Information Infrastructure and Networking Symposium (GIIS), 2015*, pages 1–6. IEEE, 2015.

[28] Bruce Hartpence and Andres Kwasinski. Considering the blackbox: An investigation of optimization techniques with completely balanced datasets of packet traffic. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 4987–4996. IEEE, 2019.

[29] Bruce Hartpence and Andres Kwasinski. Fast internet packet and flow classification based on artificial neural networks. *IEEE Trans. Southeastcon*, 2019.

[30] Bruce Hartpence and Andres Kwasinski. Combating tcp port scan attacks using sequential neural networks. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 256–260. IEEE, 2020.

[31] Bruce Hartpence and Andres Kwasinski. A convolutional neural network approach to improving network visibility. In *2020 29th Wireless and Optical Communications Conference (WOCC)*, pages 1–6. IEEE, 2020.

[32] L Todd Heberlein, Gihan V Dias, Karl N Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings. 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 296–304. IEEE, 1990.

[33] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14, 2012.

[34] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. Weka: A machine learning workbench. In *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, pages 357–361. IEEE, 1994.

[35] H Günes Kayacik, A Nur Zincir-Heywood, and Malcolm I Heywood. Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets. In *Proceedings of the third annual conference on privacy, security and trust*, 2005.

[36] Minsu Kim and Alagan Anpalagan. Tor traffic classification from raw packet header using convolutional neural network. In *2018 1st IEEE International Conference on Knowledge Innovation and Invention (ICKII)*, pages 187–190. IEEE, 2018.

[37] Seong Soo Kim and AL Narasimha Reddy. A study of analyzing network traffic as images in real-time. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, volume 3, pages 2056–2067. IEEE, 2005.

[38] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] Wei Li, Kaysar Abdin, Robert Dann, and Andrew Moore. Approaching real-time network traffic classification. Technical report, 2013.

[40] Hyun-Kyo Lim, Ju-Bong Kim, Joo-Seong Heo, Kwihoon Kim, Yong-Geun Hong, and Youn-Hee Han. Packet-based network traffic classification using deep learning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 046–051. IEEE, 2019.

[41] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017.

[42] Steve McCann. Tcpdump. 2001.

[43] Albert Mestres, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero, David Meyer, Sharon Barkai, Mike J Hibbett, et al. Knowledge-defined networking. *ACM SIGCOMM Computer Communication Review*, 47(3):2–10, 2017.

[44] Ang Kun Joo Michael, Emma Valla, Natinael Solomon Neggatu, and Andrew W Moore. Network traffic classification via neural networks. Technical report, University of Cambridge, Computer Laboratory, 2017.

[45] Andrew Moore, Denis Zuev, and Michael Crogan. Discriminators for use in flow-based classification. Technical report, 2013.

[46] Alexander I Nesterov. On angular momentum of gravitational radiation. *Physics Letters A*, 250(1-3):55–61, 1998.

[47] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit.* Elsevier, 2006.

[48] Joseph Orero, Nelson Ochieng, Mwangi Waweru, and Ateya Ismail. Detecting scanning computer worms using machine learning and darkspace network traffic. 2017.

[49] Satyendra Kumar Patel and Abhilash Sonker. Rule-based network intrusion detection system for port scanning with efficient port scan detection rules using snort. *International Journal of Future Generation Communication and Networking*, 9(6):339–350, 2016.

[50] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open vswitch. In *NSDI*, volume 15, pages 117–130, 2015.

[51] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

[52] Soujanya Poria, Haiyun Peng, Amir Hussain, Newton Howard, and Erik Cambria. Ensemble application of convolutional neural networks and multiple kernel learning for multimodal sentiment analysis. *Neurocomputing*, 261:217–230, 2017.

[53] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[54] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[55] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[56] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proceedings of the IEEE international conference on neural networks*, volume 1993, pages 586–591. San Francisco, 1993.

[57] Markus Ring, Dieter Landes, and Andreas Hotho. Detection of slow port scans in flow-based network traffic. *PloS one*, 13(9):e0204507, 2018.

[58] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Flow-based benchmark data sets for intrusion detection. In *Proceedings of the 16th European Conference on Cyber Warfare and Security (ECCWS)*, pages 361–369. ACPI, 2017.

[59] David Rolnick and Max Tegmark. The power of deeper networks for expressing natural functions. *arXiv preprint arXiv:1705.05502*, 2017.

[60] Henning Schulzrinne, Steven Casner, R Frederick, and Van Jacobson. Rfc3550: Rtp: A transport protocol for real-time applications, 2003.

[61] Guo-lin Shao, Xing-shu Chen, Xue-yuan Yin, and Xiao-ming Ye. A fuzzy detection approach toward different speed port scan attacks based on dempster–shafer evidence theory. *Security and Communication Networks*, 9(15):2627–2640, 2016.

[62] Nathan Shone, Tran Nguyen Ngoc, Vu Dinh Phai, and Qi Shi. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1):41–50, 2018.

[63] Edgard Silva, Leandro Galvão, Edjair Mota, and Yuzo Iano. Mean opinion score measurements based on e-model during a voip call. In *The Eleventh Advanced International Conference on Telecommunications*, 2015.

[64] Avinash Sridharan, Tao Ye, and Supratik Bhattacharyya. Connectionless port scan detection on the backbone. In *2006 IEEE International Performance Computing and Communications Conference*, pages 10–pp. IEEE, 2006.

[65] Stuart Staniford, James A Hoagland, and Joseph M McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1-2):105–136, 2002.

[66] Nakia Stringfield, Russ White, and Stacia McKee. *Cisco Express Forwarding*. Pearson Education, 2007.

[67] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[68] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[69] T Tang, Syed Ali Raza Zaidi, Des McLernon, Lotfi Mhamdi, and Mounir Ghogho. Deep recurrent neural network for intrusion detection in sdn-based networks. In *2018 IEEE International Conference on Network Softwarization (NetSoft 2018)*. IEEE, 2018.

[70] Gavin Watson. A comparison of header and deep packet features when detecting network intrusions. Technical report, 2018.

[71] Wenqi Wei, Ling Liu, Margaret Loper, Ka-Ho Chow, Emre Gursoy, Stacey Truex, and Yanzhao Wu. Cross-layer strategic ensemble defense against adversarial examples. In *2020 International Conference on Computing, Networking and Communications (ICNC)*, pages 456–460. IEEE, 2020.

[72] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[73] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[74] Zhiding Yu and Cha Zhang. Image based static facial expression recognition with multiple deep network learning. In *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*, pages 435–442. ACM, 2015.

[75] Shahrzad Zargari and Dave Voorhis. Feature selection in the corrected kdd-dataset. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference on*, pages 174–180. IEEE, 2012.

[76] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[77] Yu Zheng et al. Methodologies for cross-domain data fusion: An overview. *IEEE Trans. Big Data*, 1(1):16–34, 2015.

[78] Michele Zorzi, Andrea Zanella, Alberto Testolin, Michele De Filippo De Grazia, and Marco Zorzi. Cognition-based networks: A new perspective on network optimization using learning and distributed intelligence. *IEEE Access*, 3:1512–1530, 2015.