UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS Honours Project
Final Paper 2024

Title: **Traffic Classification for Community Networks using Deep Learning**

Author: Boitumelo Mokoka (MKKBOI005)
Project Abbreviation: DL4WCN
Supervisor: Josiah Chavula

Table 1: Mark Allocation

Category	Min	Max	Chosen
Requirement Analysis and Design	0	20	0
Theoretical Analysis	0	25	5
Experiment Design and Execution	0	20	20
System Development and Implementation	0	20	0
Results, Findings and Conclusions	10	20	20
Aim Formulation and Background Work	10	15	15
Quality of Paper Writing and Presentation	10		10
Quality of Deliverables	10		10
Overall General Project Evaluation	0	10	0
Total Marks		80	

Traffic Classification for Community Networks using Deep Learning

Boitumelo Mokoka
University of Cape Town
MKKBOI005@myuct.ac.za

ABSTRACT

Network Traffic Classification is a fundamental task used to analyse and optimise network performance, especially in resource constrained environments. Efficient and accurate classification is key because it can enhance Quality of Service (QoS) by implementing traffic engineering techniques, such as optimal bandwidth allocation prioritization of critical data flows. This study addresses the problem of real-time network management within Community Networks, which often face resource limitations. Efficient traffic classification is crucial for ensuring the quality of these networks, particularly in resource-constrained environments. Due to the increased complexity and encryption level of network traffic, traditional classification methods have become inadequate which is why we propose Deep Learning as an approach . We focus on the exploration of Stacked Auto-Encoders (SAEs) when combined with Recurrent Neural Networks (RNNs) to develop a computationally efficient and accurate model capable of real-time traffic classification. By comparing Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) across different Auto-Encoder architectures we aim to find the optimal balance between computational efficiency and classification accuracy. This research evaluates how these hybrid architectures perform in resource-constrained scenarios, providing insights into their potential application for enhancing QoS.

KEYWORDS

Deep Learning, Netwokrs, Traffic Classification, Stacked AutoEncoder, Recurrent Neural Networks

1 INTRODUCTION

Network Traffic Classification can be used to significantly improve the quality of community networks in many ways. In this paper we explore how deep learning techniques can be harnessed for this purpose. Traffic classification has the potential to enhance many network capabilities like resource allocation, intrusion detection and malware detection through traffic engineering techniques, also known as Quality of Service (QoS) techniques [32]. While services like video and VOIP (voice over internet protocol) require faster transmission speeds, text-based services are able to perform adequately with lower transmission speeds. Network operators and QoS devices are able to optimize bandwidth in a network by assigning different priorities to different types of services [14].

This study is performed in the context of Community networks. Community networks are community-led projects aimed at providing internet connectivity in under served areas. These networks are often setup, maintained and funded by a local community or non-profit organisation and often face resource constraints [3]. QoS is essential in networks with limited resources in order to guarantee network performance according to different application requirements. In resource-constrained community networks, QoS can prioritize critical traffic flows such as educational file transfers and VOIP to improve the overall service quality. By utilizing classified network traffic, QoS techniques optimize internet connectivity and enhance the end-user experience. [1, 17]. Accurately classified traffic also helps QoS devices perform dynamic bandwidth optimization, by applying different forwarding priorities, while security devices use this information to implement traffic policing and intrusion/malware detection [21].

Overtime the nature of internet traffic has evolved and the increased complexity and level of encryption have subsequently decreased the effectiveness of traditional classification techniques. Modern encryption procedures use pseudo-random techniques that scramble any discriminatory patterns in the network traffic, this increased complexity has significantly decreased the classification accuracy of statistical techniques [26]. Some proposed methods such as port-based methods and deep packet inspection struggle to deal with newer traffic characteristics (such as dynamic port allocation and tunneling) that have been integrated for improved security [19]. Machine Learning techniques have been proposed for this task because they do not rely on port numbers or package content but rather the characteristics of the packages[27].

Most recent research on the topic has found deep learning models to be the most accurate when it comes to classifying network traffic [8, 16, 23, 24, 27**? ?**]. While Deep Learning techniques require more computational resources that traditional techniques, their superior accuracy and ability to extract distinguishable features from extremely complex dataset make them a viable candidate. These models are able to learn patterns in raw data, without the need for packet inspection or human-engineered feature extraction, making them more adaptable to evolving network traffic characteristics [16]. It is also worth noting that deep learning models have been shown to perform better on larger datasets and this characteristic makes them ideal for traffic classification because networks tend to have extremely large amounts of raw data [24]. The aim of this study is to produce a machine learning model that is accurate and efficient enough to be used for real-time network traffic classification. These characteristics warrant a thorough investigation of the trade-off between efficiency and performance to determine the suitability of Deep Learning models for resource constrained community networks.

Hybrid Deep learning models combine different types of networks in a sequential manner. These models have been shown to be extremely effective by recent pieces of research due to their ability to leverage the abilities of multiple architectures [8, 27]. In this research we explore the use of Stacked Auto-Encoders (SAEs) for unsupervised dimensionality reduction and Recurrent Neural Networks (RNNs) for the task of classification. We benchmark these models and compare their performance to those of Weisz et al. [29] who addressed the same problem in 2021 using a Convolutional Neural Network (CNN). Additionally we compare the Encoders and classifiers with traditional statistical techniques such as Principle Component Analysis and Random Forest classifiers.

The objectives of this research are to produce a computationally efficient model with a high classification accuracy using the deep learning techniques mentioned above. We compare our implementation with that of Weisz et al. [29], who addressed the same problem using a 2D-CNN. Our aim is to improve on the work of Weisz et al. [29] by producing a model which is more suitable for real-time classification in a resource constrained environment, by improving on the computational efficiency of the model using an alternative approach.

The rest of this research is structured as follows. In Section 2 we provide the context of our research problem as well as a brief description of the deep learning techniques which we investigate. In Section3 we present a summary of the state of research into our research problem. In Section 4 we describe our approach, preprocessing pipeline and model design methodology. We describe exactly how we conducted our experiments in Section 5 and in Section 6 we discuss the results from these experiments. Finally we conclude the research with a summary of our findings in Section 7.

2 BACKGROUND

An overview of community networks and a summary of popular approaches to deep learning are presented here in the context of our classification task.

2.1 Community Networks

Community Networks are community-led projects which aim to establish accessible and affordable internet connectivity in places where these services are not available, they act as a feasible alternative to commercial internet service providers [3]. Community networks are often organized, funded by and maintained by citizens and charitable organisations which is why they often face resource constraints than traditional ISP networks [18]. These networks make internet access more affordable by aggregating the costs of internet connectivity, allowing residents of the community to purchase internet bundles at bulk rates [13]. Community networks, also known as "bottom-up networks", have been a formidable force in the battle against the digital divide with hundreds already established around the world, many of which provide services such as internet connectivity, VOIP, media streaming, messaging and file sharing [3]. When managing these kinds of networks it is essential that the scarce resources are managed efficiently in order to provide a good user experience which is why QoS techniques and, consequently, network traffic classification, are vital for community networks.

This research is performed in the context of the iNethi Community Network [13], based in Ocean View,a township located in the Southern Peninsula of Cape Town, South Africa. The network traffic dataset is sourced from this network and our aims are set with this community network in mind. The iNethi community network makes use of a local server connected to WiFi access points as well as a wireless mesh that combines WiFi and TV white spaces (TVWS) radios [13]. The aim of this research is to produce a model which can comfortable run on this mesh architecture which is why our focus is on computational efficiency.

2.2 Traffic Classification

A large reason why Deep Learning has become popular for traffic classification is because many classical methods for the task have become ineffective due to the evolving nature of internet traffic. The growing demand for privacy and security has led to a significant rise in encryption and anonymization in networks, making it difficult for traditional methods to find the discriminatory patterns necessary for accurate classification [27].

Traditional traffic classification technologies consisted of three popular approaches which each rose to popularity and eventually fell out of favour as the nature of internet traffic evolved. The oldest and simplest method is Port based classification. Port based methods were once used to extract identifiable patterns in traffic because certain port numbers were assigned by the Internet Assigned Numbers Authority for certain applications and protocols[8]. This technique has been found to struggle with new traffic characteristics such as port obfuscation and dynamic port allocation, which were introduced to improve the security of network traffic. Additionally some network services use techniques such as tunneling and anonymization to hide port numbers. Anonymization and tunneling protocols hide connection metadata (IP addresses, port numbers etc.)to ensure user privacy [19]. An alternative method is the payload-based Deep Packet Inspection (DPI) approach, which was proposed to classify traffic based the contents of the package. This method made use of predefined patterns to identify signatures and patterns from information available in the application layer of the packets [16]. This method has become obsolete due its extremely high processing overhead and inability to handle encrypted traffic [19].

In order to classify traffic without the use of deep packet inspection it is necessary to utilize features available at an application layer level, such as packet sizes, arrival-times and number of bytes. This way it is possible to perform the classification task without inspecting or relying on the contents of the package [8]. Statistical and shallow machine learning techniques make use of these features using techniques such as decision trees, Multi-Layer Perceptions and Support Vector Machines to classify the traffic. While these statistical techniques are popular, they struggle to effectively find identifiable patterns in network traffic which is why Deep Learning is often proposed as a solution [8, 16]. In this paper we investigate the application of deep learning techniques for package-based classification, which similarly uses patterns and characteristics drawn from the application layer to distinguish different transmission protocols from each other.

2.3 Deep Learning Techniques

This research employs a range of deep learning and statistical methods. In this section we present the theoretical underpinnings of deep learning and outline the functioning of the various techniques.

2.3.1 Deep Learning. Neural Networks (NNs) are supervised machine learning methods inspired by the functionality of the human brain. The basic unit of a NN is a neuron, and during training the NN uses algorithms like back propagation to adjust the connections between these neurons to better suits its task [19]. The learning process is used to approximate a mapping (optimal set of weights) that is best suited to solve a problem (e.g. a classification problem) [16]. Deep Learning is a broad term used to describe NNs with multiple complex layers. This approach has demonstrated proficiency in identifying patterns that traditional machine learning approaches struggle with [22].

A common approach to deep learning is to utilize Hybrid networks. Hybrid networks combine different generative and descriptive models to harmonise supervised and unsupervised learning [24]. Often times an unsupervised encoder is used with a classifier network for increased accuracy and efficiency. D'Angelo et al. [8] compared standalone deep learning architectures with their hybrid counterparts, discovering that hybrid networks significantly enhance performance. In the following sections we describe the different deep learning units which we use in our exploration of hybrid networks.

2.3.2 Auto Encoders. An Autoencoder (AE) is an unsupervised deep learning technique which aims to encode and then reconstruct the input at the output layer while minimizing the reconstruction error [16]. The goal of the AE is to automatically capture the relevant features from an input and represent them in a latent space with lower dimensionality. During training their ability to encode distinguishing features is measured and adjusted by the decoder's ability to reconstruct the original input, here the original input is compared to the reconstructed value using a loss function that measures reconstruction error [19]. The technique uses back propagation to train its parameters. Once their ability to encode and decode data is validated they are used for automatic feature extraction. The encoding part of the AE is then used to produce compressed data representations containing the discriminating features necessary for classification [2]. Often times AEs are used sequentially in hybrid networks to perform feature extraction while another model uses the compressed dataset to perform classification with improved accuracy [8]. It is also worth noting that while there are a set of statistical encoder that are designed to extract important features from input, these techniques are limited to linear mappings. The AE is able to represent input using non-linear combinations of features which makes it more applicable for extremely complex datasets [8].

The stacked autoencoder is a deep learning technique that consists of several auto-encoders which are combined using greedy layer-wise training. A stacked auto-encoder (SAE) is a more complex version of this procedure in which several auto encoders are combined and trained. The stacking of multiple auto encoders allows the model to learn more complex patterns in the dataset, each layer is trained separately so that the network can learn the patterns in a hierarchical fashion with each layer capturing increasingly complex features[16]. SAEs can either be trained in an end-to-end fashion or in a greedy layer-wise fashion. The greedy Layer-wise fashion, proposed by Bengio et al. [4], involves isolating and training each layer individually by freezing all other layers. This technique is presented in Figure 1 and was found to be effective by Lotfollahi et al. [16].

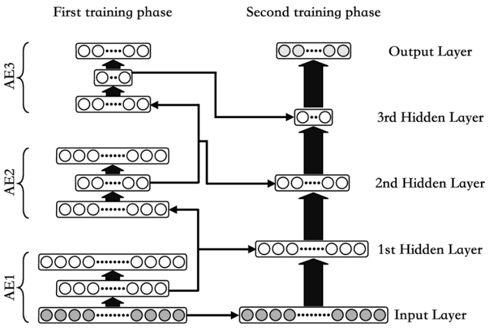

Figure 1: Greedy layer-wise SAE [16]

In this research we investigate the use of four autoencoder variations: a shallow Pure autoencoder, a deep autoencoder, a greedy layer-wise trained stacked autoencoder and finally a sparse autoencoder. The first three architectures are used for feature extraction while the Sparse AE is used for standalone classification. Additionally we compare these autoencoder architectures with the Principle Component Analysis (PCA) Encoder. PCA is a popular technique for linear dimensionality reduction. It works by finding a linear subspace of lower dimensionality while maintaining most of the variability of the input data [28]. This benchmark Encoder is introduced to compare the non-linear dimensionality reduction ability of the autoencoder with the linear functionality of the PCA.

2.3.3 Recurrent Neural Networks. Recurrent Neural Networks (RNN) are deep learning networks, originally proposed for Natural Language Processing, which are able to extract temporal relationships in data. RNNs use feedback connections to do sequence processing which allows the networks to do temporal processing and sequence learning [24]. This attribute has made them very popular for use in time-series data prediction and classification [30]. The building block of an RNN is a memory-unit that contains feedback connections, allowing the network to use information from previous inputs as context for their current input. It must be noted that standard RNNs have the issue of vanishing and exploding gradients which makes learning long-term dependencies difficult. Two variations of the RNN are the LSTM (Long Short Term Memory) and the GRU (Gated Recurrent Unit) which are described in more detail below, these two variations are improved versions of the RNN which address the fragile gradients and have been noted to perform better in real-world applications[24]. The behaviour of network traffic often contains temporal dependencies and sequential patterns, we aim to leverage the RNNs ability to capture these temporal dependencies to learn relevant features from raw network traffic data. The standard RNN architecture is visualized in Figure 2 on the left-most side.

2.3.4 Long Short Term Memory. The Long Short-Term Memory (LSTM) model was first proposed by Hochreiter et al.. [11] to solve the fragile gradient issue found in the standard RNN. The LSTM addressed the long-term dependency problem by controlling when

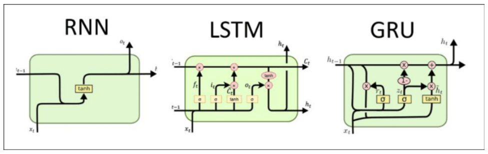

Figure 2: RNN Architectures
[25]

memory is stored and removed from the memory unit. This mechanism is implemented using an "Internal Cell State" and three gates: the Input, Output and Forget gates. The Input gate determines how much input is stored in the cell state, the forget gate determines how much past information should be forgotten and the output gate controls how much data from the cell state should be provided as output [8]. The LSTM Cell architecture is visualized in the center of Figure 2. Here it is visually evident that the LSTM implements a lot of additional complexity to the traditional RNN cell structure.

2.3.5 Gated Recurrent Unit. The Gated Recurrent Unit (GRU) is another popular variation of the RNN which uses gating methods to control information flow between cells in the network. It works extremely similarly to the LSTM but has only two gates (reset and output) instead of 3. Additionally the GRU exposes its entire cell state each time instead of using a mechanism to control the degree to which its state is exposed [7]. The GRU's simplified structure offers comparable performance to the LSTM while being significantly faster to compute [24]. It is visualized in Figure 2 on the right-most side.

2.3.6 Convolutional Neural Network. Another Deep Learning model worth mentioning is the Convolutional Neural Network (CNN). Although this study does not investigate the CNNs ability to classify traffic, Weisz et al.'s 2D CNN model [29] is used a benchmark throughout experimentation and testing. Additionally the CNN has been use extensively for the task of network traffic classification by many recent studies (see [8, 15, 16].

The CNN is a Deep Learning architecture design for image-based applications, this design has made it extremely effective at extracting spacial features in unstructured data. In this network each layer performs convolutional operations on the preceding layer to produce a feature vector on the succeeding layer [16]. These convolutional operations are combined with non-linear activation functions to learn complex features in the data. In deep learning the early layers are able to extract simple features while the deeper layers hierarchically extract very abstract features from the input [8].

2.3.7 Ensemble Learning. In this study we explore the application of RNNs for classification and compare their performance to a Random Forest Classifier. Random Forest is an effective ensemble learning technique that has been widely used in network traffic classification. Ensemble learning involves training multiple (weak) classifiers and then aggregating the results either by voting or averaging the answers across the classifiers. The random forest is a specific ensemble method belonging to the bootstrapping ensemble type. It works by forest of decision trees with each tree being trained on a random subset of the data and features. This approach reduces the bias, variance and risk of over fitting. [19]

3 RELATED WORK

Given its significance for quality network connectivity, QoS and, consequently, network traffic classification have been extensively explored by both researchers and ISPs [8]. Recent studies have consistently demonstrated the improved performance of deep learning models when compared with traditional approaches [19]. In the following section we will be providing an overview of the current state of research into our research problem.

Comparison of research results is difficult because each study tends to use different performance metrics and different datasets. Luckily many popular works employ the publicly available ISCX VPN-non VPN dataset which was developed by Draper-Gil et al. [10] during their research on encrypted traffic characterization. This dataset consists of labeled network traffic in PCAP format and has been used by researchers such as Lotfollahi et al. [16] and Izadi et al. [23].

Below we have chosen to summarize works which we see as more relevant to our study and context. In order to provide a valid comparison in our context we report on studies which consider the network traffic at packet-level instead of flow-level. While some papers to investigate the use of flow-based classification, we decided to omit this approach. Package based classification is more suitable for real-world implementation because it does not require packets to be grouped into flows before being classified.

A prominent piece of research, conducted by Lotfollahi et al. [16], utilized CNN and Sparse AE-based configurations for network traffic classification. Their deep learning framework, named Deep Packet, is able to identify encrypted traffic and categorize the traffic with a recall of 0.94. At the time of publishing this implementation outperformed all of the proposed methods on the ISCX dataset. This study achieved a predictive accuracy of 98% and 96% for the 1D-CNN and Sparse AE respectively. Their SAE architecture was trained using 5 fully-connected layers (400, 300, 200, 100, and 50 neurons respectfully). They also employed a dropout rate of 0.05 between each layer to prevent over fitting. Lotffollahi et al.'s [16] model and experiment implementation serves as a template for our own Sparse AE implementation in this research.

D'Angelo et al. [8] investigated a wide range of hybrid deep learning networks for network traffic classification, the literature compared the use of traditional approaches with deep learning approaches and even took a step further by comparing standalone deep learning approaches with their hybrid counterparts. While their CNN-NN and LSTM-NN models performed sufficiently, the introduction of an SAE increased the performance significantly. The accuracy for the CNN-SAE and LSTM-SAE hybrids increased by 9% and 8%, with respect to their counterparts, respectfully. Their best performing model was a stacked CNN-LSTM-SAE-NN, which scored near 100% on all metrics. It is also worth noting that they used Amazon Web service machines to perform the task, 4 instances with 4 vCPU Intel Broadwell E5-2686v4 @ 2.3 GHz, and 16 GB RAM were utilised. This is a relatively low-resource specification and is worth noting because of the resource constraints on which we will be testing in our implementations.

Rezaei et al. [21] provide a broad overview of Deep learning for encrypted traffic classification in their research. In this work they provide an evaluation of the applicability of deep learning for real time classification and raise some points of concern. Firstly the fact that network applications are always evolving warrants the periodic evaluation and update of deep learning models. Some proposed solutions involve detecting unlabeled clusters and then labelling them using unsupervised learning techniques such as K-Means clustering. Another valid argument against deep learning's applicability to the research problem is that extremely large volumes of data are required in order to utilize Deep Learning. Izadi et al. [23] suggest the investigation of data fusion, the combination of different datasets from different sources. When combined with randomized sampling this technique will produce models which are more adaptable and robust to different types of traffic.

Recent studies consistently demonstrate the improved performance of deep learning methods over traditional approaches. Both Lotfollahi et al. [16] and D'Angelo et al. [8] were able to utilise Autoencoders to achieve a classification accuracy of above 95% for their respective models. While valid concerns for the applicability of deep learning models for real-time classification are also raised, countermeasures have been proposed too. The research over the last decade provides a compelling case for Deep Learning as a candidate for use in real-time applications.

4 DESIGN AND IMPLEMENTATION

4.1 Approach

Our dataset is sourced from the Ocean View Community network [13] and the preprocessing of this data will take place on our machines using two open source-libraries pkt2flow and nDPI. We then used this preprocessed data to train and test our deep learning models. TensorFlow will be used as the primary machine learning library for the implementation of our deep learning models, and the Keras API [6] (which is integrated with TensorFlow) will be used to define the model architectures and perform hyperparameter tuning. We outsourced hardware by utilizing Google Colab's online environment to run our models on external GPU servers. Google Colab is a paid service which gives you access to cloud GPUs for efficient model training and testing.

4.2 Network Traffic Data

We will be using a dataset that has been collected from the Ocean View Community network in Cape Town [13]. The raw network data is a set of PCAP files that have been collected from the gateway of the community network and consists of traffic flowing between the internet and the network from February 2019 until May 2019. The PCAP files are stored in a data repository at the University of Cape Town, through which we have been granted access to the data.

4.3 Preprocessing

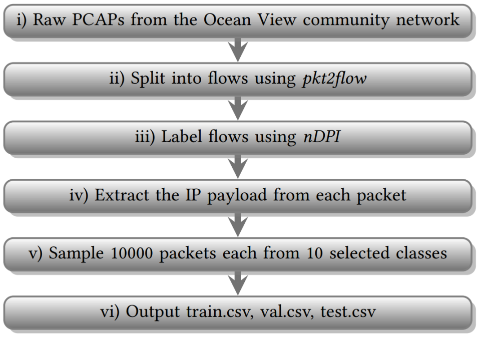
i) Raw PCAPs from the Ocean View community network
ii) Split into flows using *pkt2flow*
iii) Label flows using *nDPI*
iv) Extract the IP payload from each packet
v) Sample 10000 packets each from 10 selected classes
vi) Output train.csv, val.csv, test.csv

Figure 3: Preprocessing pipeline

To make the training and testing of our deep learning models as streamlined as possible we produced a training, testing and validation set in CSV format. The final output of our preprocessing pipeline is a large labelled dataset containing 10,000 samples of each category. This pipeline was achieved using a combination of open source libraries and python scripts to convert wireshark capture PCAP files into labeled Byte data. This process is expanded upon in the following sections and visualized in a flow diagram in figure 3.

4.3.1 Packet Labelling. In order to produce a labelled dataset suitable for deep learning training, we utilized a combination of python scripts and open-source libraries. First, we split the raw PCAP data into individual flows using the *pkt2flow* [5] library, this step was necessary in order to label each flow. Next we used the *nDPI* library to label the flows, this library assigns each packet in a given flow to that flow's label. Once these two steps were complete we extracted 1480 bytes of the IP Payload using the python package *scapy*. Within a dataset the source and destination IP addresses are unique for each application, we chose to omit the IP address field in the preprocessing phase to improve the model's ability to generalize to unseen data. In this step only the raw bytes of the IP payload are extracted and used for training. Packets with less than 1480 are filled using zero-paddding and packets that do not contain a payload are completely discarded, this way we ensure that the final output vector is the same for each packet.

4.3.2 Sampling and Balancing. Once a large "data.csv" file is produced, we sampled and balanced the dataset to ensure that each category is sufficiently represented by removing categories that are underrepresented and under-sampling categories that are over represented. Our motivation for balancing the datasets is motivated by studies showing extremely unbalanced datasets to have adverse effects on the deep learning training process (e.g. [19]). We randomly split the dataset into 3 parts, 64% of the samples were used for training, 16% of the samples were used for validation during training and the final 20% of samples were used for accuracy testing and model evaluation. This train-test-validation split is common in statistics because it offers a balance between sufficient training data, adequate testing data and enough validation data for hyperparameter tuning. A histogram displaying the final dataset and the train-testing-validation distribution is shown in Figure 4. This under-sampling method and set-balancing ratio was adopted by Lotfollahi et al. [16] in their implementation.

4.4 Model Design

We considered three autoencoder variations, two RNN classifiers, a 1D-CNN classifier and 2 statistical methods in a grid search for an

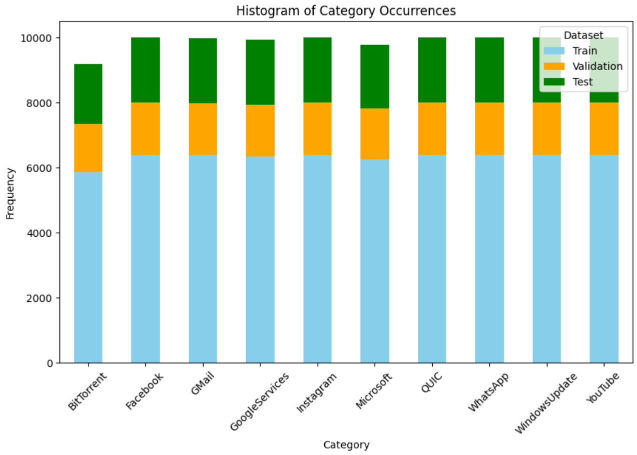

Figure 4: Histogram of Balanced Dataset

optimal hybrid model. Additionally we bench marked our candidate models against the 3 layer 2D-CNN classifier which was found to be most effective by Weisz et al. [29].

Our three AE variations were the shallow AutoEncoder, the Stacked AutoEncoder and the PCA Encoder. The Shallow-AE was a simple 3 layered model, consisting of equal input and outputs layers with a 100 dimension latent space between them, after training the output of this model was a simple 2-layered encoding model. The Stacked-AE consisted of 3 layers that were each trained separately using greedy layer-wise training. Once trained, the Stacked-Encoder was made up of 3 layers, containing 1480, 400 and 100 neurons respectively.

During the the first experiment we perform a grid search by connecting all three AutoEncoder variations to all three classifiers in order to form 9 hybrid models in total. The two RNN models (RNN and GRU) were designed with the exact same architecture to compare their classification ability. This architecture consisted of one 16-cell layer, one 8-cell layer (of the respective RNN cell types) and a softmax classifier.

During Benchmarking the chosen deep learning models were aligned to similar parameter sizes. The 2D-CNN consisted of three layers (64,16,16) to match the number of parameters contained in the AE-Random Forest model.

4.5 Model Training

During experimentation we standardized the training procedure across all models to ensure the experiments were performed in a reproducible and controlled environment.

While training the classifiers we use the validation data set to perform k-fold cross validation at each epoch, this process is used to optimise the model parameters. This process is widely used in hyperparameter tuning and allows a additional adjustment of hyperparameters at the end of each epoch to esnure the model does not overfit to the training dataset [31].

During training we will train our SAEs and RNN classifiers separately: the SAEs will be evaluated on their ability to encode and reconstruct just the input data, this unsupervised pre-training aims to approximate a function which extracts complex features from the input data. The encoding parts of these SAEs is then extracted and connected to the classifier, thus the RNN classifiers are trained on encoded datasets after feature extraction. Each of the SAE networks were trained using the *Adam* optimizer with mean squared error used as their loss function. After this the RNN classifiers were trained in tandem with the pre-trained SAE encoders with categorical cross-entropy as their loss function, this training also used the Adam optimizer.

Additionally many training techniques are utilized to improve the training process: early-stopping is a technique which stops the training process when the loss function remains unchanged for several iterations. Dropout is a methodology to reduce over fitting of a network, it works by randomly nullifying a certain portion of the nodes in the input and hidden layers at each iteration during training. We utilized these additional techniques during the second stage of experiments, once we had settled on which autoencoder architecture to implement. The specifics of the dropout and regularization parameters are described in Section 5.

4.6 Evaluation Metrics

The predictive accuracy of the different models will be evaluated using a set of statistical evaluation metrics: namely the accuracy, precision and recall. The statistical evaluation was implemented using Python's Scikit Learn library. Precision measures the accuracy of the positive predictions while recall (also known as sensitivity) measures the ability of the classifier to identify positives. A high precision means that if the model predicts a class its is very likely to be correct, and a high recall values means that the model is good at identifying all relative classes [9]. The formulae for Precision and Recall is as follows:

$$Precision = \frac{True\ Positives}{True\ Positives +\ False\ Positives}$$

$$Recall = \frac{True\ Positives}{True\ Positives +\ False\ Negatives}$$

We decided to omit the calculation of the F1 score, which measures a weighted accuracy across all of the categories, because we performed dataset balancing beforehand.

We measure the computational efficiency of each model using packets classified per second and number of parameters. The speed of each model is measured in **packets classified per second** and Python's Time library is used to measure this during testing. The formula for this calculation is as follows.

$$PacketsPerSecond = \frac{Number\ of\ Packets}{Classification\,Time}$$

Weisz et al. [29] estimated the Ocean View Community network to process approximately 10 000 packets per second on average, because they have a network connection speed of 10mbps. This number will be used when evaluating our models for real-time classification.

The number of parameters in a deep learning model, represents the number of weights and biases that the model learns during training. For SAEs and CNNs this value is computed using the number of connection between each layer in the model. For random forest structures the parameters are calculated using the structure of the trees (number of nodes, splits and leaf values) and then this value is multiplied by the number of decision trees in the forest. The number of parameters determines the amount of memory the model will consume when running. While an increased number of parameters does tend to improve accuracy, large models require more computational resources.

5 EXPERIMENT DESIGN

In this section we describe the methodology and motivation for our experiments to investigate and evaluate the suitability of a SAE-RNN for the classification task. Our primary goal was to strike a balance between computational efficiency and accuracy while comparing our proposed model to the benchmarked 2D-CNN and MLP classifiers. To achieve this, we employed a two-phase approach :

(1) **Model Selection:** In the initial phase of this experiment we assessed the encoding capabilities of the three Autoencoder (AE) architecutre and compared the accuracy of different RNN classifiers across all there AE variations. This phase aimed to characterize the strengths and weaknesses of each variation and identify a candidate combination of AE and RNN that could compete with the benchmark performance of the 2D CNN.

(2) **Prediction Speed vs Number of Parameters:** To evaluate the scalability of our selected AE variant we conducted an experiment to see how its performance varied with model complexity. This experiment was designed to determine the optimal balance between classification accuracy and speed for our task.

(3) **Benchmarking:** Here we compared our chosen candidate models against the established benchmarks, namely the 2D-CNN, to assess its performance in terms of both accuracy and computational speed. In this phase we will also be testing the effect of different resource constraints on predictions speed: Testing the chosen model on different devices with different resource constraints (RAM size, CPU speeds etc)

To determine the most suitable autoencoder variation for our task and dataset we conducted an additional comparative analysis between the deep, shallow and layer-wise AEs. The deep autoencoder consisted of 4 encoding layers and 4 decoding layers such that the encoder would reduce the dimensions sequentially in the following order: 1480,400,200,100. The shallow autoencoder consisted of 3 simple layers (1480, 100, 1480). Both the shallow and deep autoencoders underwent the same end-to-end training with all encoding and decoding layers partaking in backpropagation adjustments at every epoch. The layer-wise autoencoder consisted of 3 encoding layers which were each trained in isolation using greedy layer-wise training. This training schedule, visualized in Figure 1 worked by training 2 different AEs, one at a time, and conjoining their respective encoding layers to form a full encoder. All three of the AEs underwent the same training conditions: using the Adam optimizer we performed 200 epochs of unsupervised AE training and 30 epochs of classifier training. A dropout rate of 0.2 was used between each layer to avoid over fitting and the relu activation function was used at each layer in order to approximate non-linear features from the input data. Each model additionally used *early stopping*, a technique that halts the training process if there is no improvement in validation accuracy over the last 5 epochs.

This first experiment functioned as a screening process to evaluate the characteristics of the different deep learning models. We chose to use a relatively short training time of 30 epochs in order to perform rapid evaluation of each model combination, using this experiment design we are able to discard model combinations that are clearly under performing and perform in-depth training and fine-tuning of the models we found to be promising.

6 RESULTS AND DISCUSSION

6.1 Model Selection

AutoEncoder Name	Average Packets Per Second	LSTM Accuracy	GRU Accuracy	Random Forest Accuracy
Shallow AutoEncoder	4203.91	0.1211	0.25366	0.5610
Layer-Wise AutoEncoder	4203.8726	0.1011	0.5971	0.5872
PCA Encoder	5784.4358	0.1011	0.3478	0.6403

Table 2: Comparison of Autoencoder Variations

The results revealed a lot about both the RNN classifiers and the AE variations. The GRU was shown to outperform the LSTM at classification accuracy across all three AE variations. This is most likely due to the reduced complexity of the GRU which allows to to avoid over fitting to the training data. Considering the limited training times the GRU and Random forest models were able to achieve an accuracy above 5% while the LSTM failed to classify above 15%. These results serve as motivation to use the GRU going forward in our model selection experiment.

While the Shallow and Layer-wise autoencoders achieved similar packets per second, the Layer-Wise AE achieved a higher overall accuracy. The fact that the shallow AE could perform similary to the Layer-Wise AE with a largely reduced complexity motivated us to use it as a candidate architecture in the following experiments. Using the results from this first screening we identified the Shallow AutoEncoder as a promising candidate for feature extraction. We then performed a scalability test on the AE architecture by measuring its performance across a range of classifier complexities. By varying the number of parameters we were able to measure at which point the applicability for real-time classification diminishes. The results of this experiment are visualized in Figure 5.

The results from this experiment exposed the poor scalability of the GRU classifier. The computational complexity of this trained classifier was by far the highest and furthermore it diminished rapidly as the number of parameters increased, scalability makes the GRU unsuitable for real-time classification. Its inability to scale implies that we are unable to increase its complexity for the sake of improved accuracy. While the CNN outperformed the deep AE-Classifier variations on both classification speed and test accuracy, the AE-Random Forest classifier managed to achieve a suitable accuracy with an improved classification speed. Based on the screening

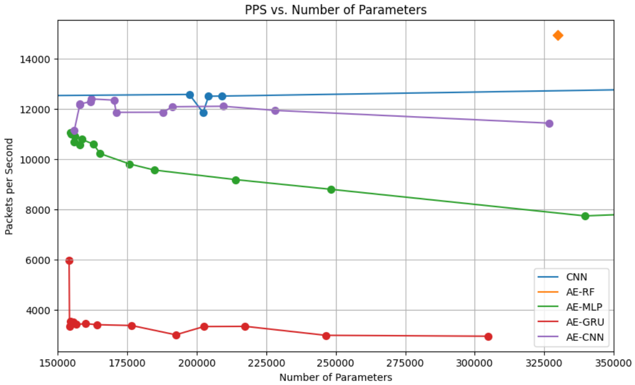

Figure 5: Packets per Second vs Number of Parameters

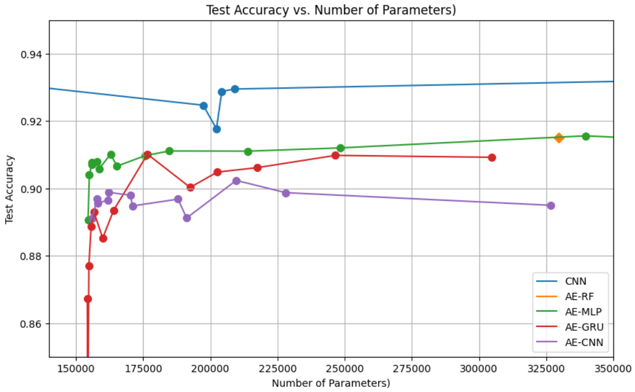

Figure 6: Accuracy vs Packets per Second

Model	Packets Per Second	Accuracy	Precision	Recall
AE - RF	10740.48	0.9151	0.9153	0.9151
PCA - RF	37977.25	0.8029	0.8029	0.8029
2D CNN	12474.39	0.919	0.9213	0.9212

Table 3: Benchmark Comparison

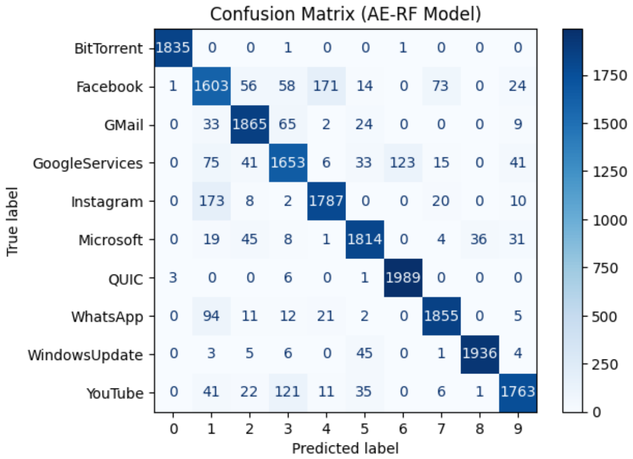

Figure 7: AE-RF Confusion Matrix

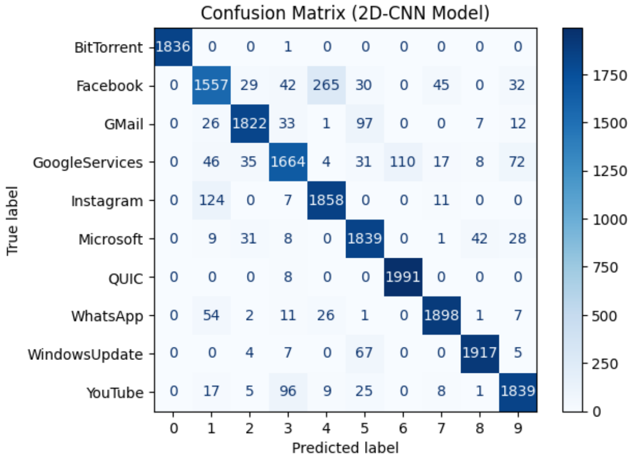

Figure 8: 2D-CNN Confusion Matrix

results we decided to use the shallow AutoEncoder when combined with the random forest classifier to benchmark against the CNN.

6.2 Benchmarking

We conducted our benchmarking using the Shallow Autoencoder - Random Forest hybrid network and compared it to a 2D-CNN with a similar number of parameters. We benchmarked their accuracy across a range of metrics and measured their classification speed. Additionally we used a PCA-RF model to compare the Autoencoders performance with a linear encoder..

While the PCA-RF model achieved the highest classification speed, it achieved the lowest accuracy across all of the metrics. This trade off between speed and accuracy was likely due to the simple linear nature of the PCA when compared with the non-linear function approximation capabilities of the AE and 2D-CNN. The linear PCA is able to classify packets at a much higher speed due to its simpler nature but this comes with a decrease in ability to identify and extract patterns in the encrypted data.

Although the AE-RF was the best performing hybrid model, the Random-Forest model could only be trained at a fixed maximum number of parameters, limiting our ability to further investigate it. For this reason we decided to investigate the AE-MLP as its number of parameters can be extended much further than 500 000. The results for this experiment are shown in figure 9.

This experiment showed the potential for the AE-MLP model as an alternative to the 2D-CNN model. When the number of parameters was extended beyong 500 000 the model was able to consistently achieve an accuracy above 92%. This experiment also showed the ideal number of parameters for this model to be around 500 000 parameters.

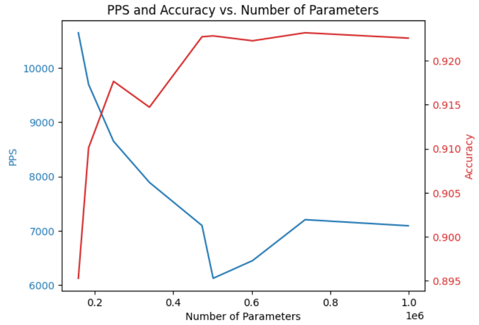

Figure 9: AE-MLP Experiemtn

7 CONCLUSION

The study demonstrated the potential for hybrid deep learning techniques for real-time traffic classification in resource-constrained community networks. By investigating different hybrid architectures, including LSTM and GRU models, this research produced a computationally efficient and accurate candidate model, capable of handling encrypted and complex network traffic. The integration of semi-supervised methods allowed the model to consistently achieve an accuracy above 92% while classifying over 7000 packets per second. The results show that the hybrid deep learning models are compatible with traditional statistical classifiers such as Random Forests. The AutoEncoder is able to effectively extract patterns in the encrypted dataset by approximating a non-linear function. These extracted patterns can be classified using much simpler classifiers such as the MLP and Random Forest Classifier.

While the 2D-CNN model performed slightly better in terms of accuracy and speed, the hybrid model's competitive performance suggests it can be a strong candidate for real-time applications. We were able to produce a candidate model which competes with the benchmark on both accuracy and predictive speed. We found that the ideal number of parameters for the AE-MLP network was 500 000 parameters, this is where the accuracy was shown to plateau and just before the computational complexity caused the classification speed to fall below 7000 packets per second.

While a high accuracy was achieved, both the benchmarked model and the AE-RF displayed a tendency to confuse social media packets. Facebook and Instagram packets were often misclassified as eachother most likely due to the nature of social media traffic. These patterns show potential to be used for zero-shot classification. For example all social media traffic could be classified as such due to their shared characteristics, such as packet sizes, transmission times, and encryption methods. This approach could be particularly beneficial in real-time scenarios where quick identification of general traffic categories is more important than distinguishing between specific services. Additionally, this method could allow for more adaptable traffic management strategies, where priority is given based on traffic type rather than individual applications, enhancing Quality of Service (QoS) in resource-constrained environments.

It is also worth noting that the AutoEncoder might be a suitable alternative to the 2D CNN due to the unsupervised learning component. Unsupervised learning could be used to update and replace the encoder when there are changes to the nature of the network traffic, unlike the end-to-end training of the CNN which would require a large dataset of labeled traffic in order to update.

7.1 Limitations

One limitation of this research was the training time and computing resources necessary to perform the experiments during investigation. Athough the use of Google Colab's accelerated GPU service greatly improved the training times, the more complex models (number of parameters above 300 000) would sometimes take hours to train and test. This placed a constraint on how many experiments we were able to run. Ideally we would have tested each model up to 2 million parameters and experimented using a wider rang of hybrid architectures. The limit on computational resources also constrained which types of networks we were able to experiment with. For example, it was not possible to experiment with a non-linear PCA encoder because the training compute time was over 5 hours due to the size of our dataset.

7.2 Future Work

Although we were able to produce a candidate model, further investigation is necessary to fully evaluate its suitability for real-world implementation. It is necessary to see how the model performs when implemented in different programming languages. This study demonstrated that the Autoencoder can be paired with simpler classifiers, this warrants investigation into the use of statistical classifiers such as the Naive Bayes and Support Vector Machine models for the task of classification.

It has been noted that similar traffic types such as social media traffic were often misclassified. Future improvements could focus on refining the model to better distinguish between highly similar traffic types or developing a hybrid approach that combines broad zero-shot classification with more fine-grained inspection where necessary.

8 RESOURCES

8.1 Software

All of the deep learning models in this research were implemented using Tensorflow's Keras library [6] and Scikit Learn [20]. Keras was used to construct train and tune the autoencoders and the RNN classifiers while Scikit Learn was used to implement the PCA encoder and the Random Forest Classifier. Scikit Learn was also used to calculate the metrics and to visualize the confusion matrices. Python's Matplotlib library [12] was used to display all of the graphs in this research.

8.2 Hardware

We used Virtual Machine on a server at the University of Cape Town's Computer Science Faculty to perform the preprocessing pipeline. We utilized Google Colab's jupyter notebook environment in order to access GPUs capable of training and testing our deep learning models. Finally, the bench marking of the models was performed on our local machines.

Boitumelo Mokoka
University of Cape Town
MKKBOI005@myuct.ac.za

9 COMPLIANCE WITH ETHICAL STANDARDS

Our use of a dataset collected from the Ocean View Community Network raises ethical concerns around the confidentiality of this data. The dataset has been ethically collected, securely stored and has been only used for academic research at the University of Cape Town. The data is securely stored on the University of Cape Town's NAS server, accessible only to authorised personnel. Our preprocessing draws the raw network traffic from the secure server using a VPN and all of our preprocessing is done on our machines. This way we are able to categorize the network traffic without revealing specific details to protect the Ocean View Community's privacy. Once preprocessed our data contains no identifiable data than can be traced back to any single person using the network, our processed data is fully anonymized and only contains the data stored in the payloads in raw numerical byte format.

REFERENCES

[1] Adedayo, A. O., and Twala, B. Qos functionality in software defined network. In *2017 International Conference on Information and Communication Technology Convergence (ICTC)* (2017), IEEE, pp. 693–699.

[2] Aouedi, O., Piamrat, K., and Bagadthey, D. A semi-supervised stacked autoencoder approach for network traffic classification. In *2020 IEEE 28th International Conference on Network Protocols (ICNP)* (2020), IEEE, pp. 1–6.

[3] Bart Braem, Christoph Barz, F. F. C. B. H. R. L. N. J. B. P. E. A. L. K. S. P. R. B. V. A. N. B. T. I. V. I. B. M. M. A case for research with and on community networks. *ACM SIGCOMM Computer Communication Review 43* (07 2013), 68–73.

[4] Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. *Advances in neural information processing systems 19* (2006).

[5] caesar0301. pkt2flow. https://github.com/caesar0301/pkt2flow, 2014.

[6] Chollet, F., et al. Keras. https://keras.io, 2015.

[7] Chung, J. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[8] D'Angelo, G., and Palmieri, F. Network traffic classification using deep convolutional recurrent autoencoder neural networks for spatial–temporal features extraction. *Journal of Network and Computer Applications 173* (2021), 102890.

[9] Gareth James, Daniela Witten, T. H., and Tibshirani, R. *An Introduction to Statistical Learning with Applications in R.* Springer, 2023.

[10] Gil, G. D., Lashkari, A. H., Mamun, M., and Ghorbani, A. A. Characterization of encrypted and vpn traffic using time-related features. In *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP 2016)* (2016), SciTePress Setúbal, Portugal, pp. 407–414.

[11] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9* (12 1997), 1735–80.

[12] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering 9*, 3 (2007), 90–95.

[13] iNethi Technologies. inethi technologies. https://www.inethi.org.za/, 2024. Accessed: March 24, 2024.

[14] Lim, H.-K., Kim, J.-B., Heo, J.-S., Kim, K., Hong, Y.-G., and Han, Y.-H. Packet-based network traffic classification using deep learning. In *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)* (2019), pp. 046–051.

[15] Lopez-Martin, M., Carro, B., Sanchez-Esguevillas, A., and Lloret, J. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access 5* (2017), 18042–18050.

[16] Lotfollahi, M., Jafari Siavoshani, M., Shirali Hossein Zade, R., and Saberian, M. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing 24*, 3 (2020), 1999–2012.

[17] Michael Finsterbusch, Chris Richter, E. R., Muller, J.-A., and Hanssgen, K. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys Tutorials 16*, 2 (2014), 1135–1156.

[18] Micholia, P., Karaliopoulos, M., Koutsopoulos, I., Navarro, L., Vias, R. B., Boucas, D., Michalis, M., and Antoniadis, P. Community networks and sustainability: a survey of perceptions, practices, and proposed solutions. *IEEE Communications Surveys & Tutorials 20*, 4 (2018), 3581–3606.

[19] Ola Salman, Imad H. Elhajj, A. K., and Chehab, A. A review on machine learning–based approaches for internet traffic classification. *Annals of Telecommunications 75* (06 2020), 673–710.

[20] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[21] Rezaei, S., and Liu, X. Deep learning for encrypted traffic classification: An overview. *IEEE Communications Magazine 57*, 5 (2019), 76–81.

[22] Roya Taheri, Habib Ahmed, E. A. Deep learning for the security of software-defined networks: a review. *Cluster Computing 26* (July 2023), 3089–3112.

[23] Saadat Izadi, M. A., and Rajabzadeh, A. Network traffic classification using deep learning networks and bayesian data fusion. *Journal of Network and Systems Management (2022)* (January 2022).

[24] Sarker, I. H. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science (2021) 2*, 420 (August 2021).

[25] Toharudin, T., Pontoh, R., Caraka, R., Zahroh, S., Lee, Y., and Chen, R.-C. Employing long short-term memory and facebook prophet model in air temperature forecasting. *Communication in Statistics- Simulation and Computation* (01 2021).

[26] Velan, P., Čermák, M., Čeleda, P., and Drašar, M. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management 25*, 5 (2015), 355–374.

[27] Wang, P., Ye, F., Chen, X., and Qian, Y. Datanet: Deep learning based encrypted network traffic classification in sdn home gateway. *IEEE Access 6* (2018), 55380–55391.

[28] Wang, Y., Yao, H., and Zhao, S. Auto-encoder based dimensionality reduction. *Neurocomputing 184* (2016), 232–242. RoLoD: Robust Local Descriptors for Computer Vision 2014.

[29] Weisz, S., and Chavula, J. Community network traffic classification using two-dimensional convolutional neural networks. In *International Conference on e-Infrastructure and e-Services for Developing Countries* (2021), Springer, pp. 128–148.

[30] Wu, Y., Tan, H., Qin, L., Ran, B., and Jiang, Z. A hybrid deep learning based traffic flow prediction method and its understanding. *Transportation Research Part C: Emerging Technologies 90* (2018), 166–180.

[31] Yann LeCun, Y. B., and Hinton, G. Deep leraning. *Nature 521* (May 2015), 436–444.

[32] Yung-Fa Huang, Chuan-Bi Lin, C.-M. C., and Chen, C.-M. Research on qos classification of network encrypted traffic behavior based on machine learning. *Electronics 10*, 12 (2021).