DEPARTMENT OF COMPUTER SCIENCE

CS Honours Project Final Paper 2024

Title: A Comparison of CNN and MLP Models for Wireless Community Networks Using Supervised and Unsupervised Learning

Author: Ryan Murphy

Project Abbreviation: DL4WCN

Supervisor(s): Josiah Chavula

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 (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	0
Total marks		**80**	

A Comparison of CNN and MLP Models for Wireless Community Networks Using Supervised and Unsupervised Learning

Ryan Murphy
University of Cape Town
South Africa
MRPRYA002@myuct.ac.za

1 ABSTRACT

Network traffic classification is a technique that has been used in a wide variety of tasks, with many different techniques used in order to classify packets or detect network intrusion. Methods such as packet inspection and port inspection used to be common but, due to encryption, have now become obsolete. The proposed solution to the encryption problem is machine learning. In this project we look at machine learning, particularly deep learning, as an accurate technique to classify network traffic. However, deep learning models are often computationally expensive, a factor we considered in our research. This paper will discuss methods for classifying network packets for a community network, which generally have less computational resources than Internet Service Provider (ISP) supported networks. This makes a model that is too computationally expensive infeasible. Therefore, besides accuracy, one of the main concerns is computational efficiency and prediction time. Fast predictions are crucial for any network classifier because if a model takes too long to predict, the data may become irrelevant. In particular, we will be looking at Multi-Layer Perceptrons (MLP), which are the most standard form of neural network. Furthermore, we will be looking at 1d and 2d Convolutional Neural Networks (CNN) which evolved from the MLP by having a 1d and 2d sliding window respectively to pick up temporal and spatial features in data. We will be comparing a regular 2d-CNN with models that have had their input feature space reduced significantly in order to make them faster while still maintaining a good accuracy level.

2 INTRODUCTION

Network traffic classification involves taking the contents of packets and classifying them into their associated type. This is beneficial because network administrators can allocate resources to traffic types that are of priority at a point in time. This is especially useful for low resource networks because they may not have enough capacity to give enough resources to all packet types and may have to allocate more resources to packet types that are a higher priority at that given point which can ensure Quality of Service (QoS). This is why they could benefit from a traffic classifier. The low resource environment is what motivates our research.

Network traffic classification techniques have had to evolve over time due to the introduction of encryption. Traditional techniques like port based classification and packet based inspection such as Deep Packet Inspection (DPI) techniques have become less popular over time. Port obfuscation has made port based classification ineffective [35] and packet inspection methods are too computationally expensive and too time consuming [7]. Machine learning has emerged as one of the most prominent techniques and in particular deep learning has shown the most effective results. Traditional machine learning techniques like K-nearest neighbours and Decision Trees have been used for traffic classification and have shown very high accuracy. However, these methods tend to rely on highly human engineered features which can increase overhead costs and reduce generalisability [2].

Deep learning, which is a subset of machine learning, has become the most popular method for packet classification. Although, deep learning models tend to have very large parameter counts, which tend to increase prediction time and computational expensiveness. The goal of this project is to find models that have the right balance of efficiency, without compromising too much on accuracy. We will be creating lightweight deep learning models to perform network traffic classification. This will consist of a 1d-CNN, a 2d-CNN and an MLP. Furthermore, we will be applying a dimensionality technique to lower computational load. This brings us to our two major research questions:

2.1 Research Questions

(1) Will our models with the dataset reduced to 100 features per record be able to achieve similar accuracies to our benchmark model, a 2d-CNN using the original dataset?
(2) Will our models with dimensionality reduction be able to outperform the benchmark 2d-CNN when it comes to number of parameters classified per second?
(3) Will our models with dimensionality reduction be able to outperform the benchmark 2d-CNN when it comes to peak memory usage when classifying a single packet on low resource architectures?

We will benchmark our models against a model created by Weisz [33] who did a similar study using a similar dataset. Weisz implemented a 2d-CNN. We have created our models to try to find one that is still accurate but more computationally efficient. We are using an unsupervised learning technique for feature reduction known as Principal Component Analysis (PCA) to reduce the input feature space, while still retaining the most relevant input features. This should result in better classification times and better computational efficiency.

3 BACKGROUND

3.1 Community Networks

Community networks are community governed and operated network infrastructures for digital communication [17]. The users are the ones who run them and manage them with funding coming from the users themselves. The purpose of these networks is to provide internet connection and accessibility to places where ISPs or governments are yet to reach. They are extremely important in low resource communities as they provide low-cost internet access. Since the people using community networks are the ones who run them, there is not as much expertise, funding or capital compared to an ISP run network. Due to the lack of resources, QoS is something that these networks absolutely require.

3.2 Challenges and Constraints

The main challenge of this project is to find solutions that are not only accurate but also are not too computationally expensive. Deep learning models tend to be the most accurate for studies like these, but are also very computationally intensive. A model that is very computationally expensive cannot work for a resource constrained environment like a community network. Therefore, in this project we will look for the right balance between accuracy and computational expensiveness. We will be testing an array of model configurations and hyper-parameters to try to find the right trade-off between accuracy and computational expense.

3.3 Machine Learning

Machine learning algorithms are algorithms that are able to learn patterns and adapt without explicit instructions. By discerning patterns, relationships, and trends within the data, machines become capable of making informed decisions and predictions in various domains [4]. This is done by feeding these models training data for the model to learn using some sort of optimisation algorithm. The model is then tested on unseen (out-of-sample) data to assess its performance after training. This unseen data is called test data. Traditional machine learning algorithms include Decision Trees, Regression models, K-nearest neighbours and others [21].

3.4 Supervised and Unsupervised Learning

Supervised, unsupervised, and reinforcement learning are the three main categories in machine learning. Reinforcement learning is not within the scope of this project; therefore, we will focus on the former two. Supervised learning involves predicting some target variable Y given a set of p predictors, $X_1, X_2, \ldots, X_p$. These include regression tasks which aim to predict a continuous target variable and classification tasks which aim to classify some discrete or categorical variable. Supervised models learn from data using a loss function which tells an optimisation algorithm how different the model's predictions are compared to the true results. An optimisation algorithm adjusts the parameters of the model based on that loss [11].

Unsupervised learning however, is intended for cases when we are not interested in predicting some target variable. Rather, the goal is to discover interesting things about the measurements on $X_1, X_2, \ldots, X_p$ [11]. Unsupervised learning primarily focuses on identifying patterns or structures in data without predefined labels [12]. The three main tasks are clustering, association rule mining and dimensionality reduction. In this project we will be using the latter, namely Principal Component Analysis, for dimensionality reduction. Dimensionality reduction is used to reduce the amount of features per record in a dataset while preserving as much of the information or variance in the dataset as possible [11].

This type of machine learning, where the data undergoes an unsupervised learning algorithm and is then fed into a supervised learning model, is known as semi-supervised learning [3].

3.5 Deep Learning

Deep learning is a subset of machine learning and can be used for supervised or unsupervised tasks. Deep learning uses artificial neural networks to learn patterns in data. These neural networks are very powerful at learning complex patterns and extracting hierarchical features in data, making them versatile and effective [16]. Graphics processing units (GPU), which are commonly used in computer graphics, are now being used in machine learning due to their high degree of parallelism [24]. Deep learning models do a lot of calculations on each piece of data fed to them and these datasets can be very large therefore, taking a lot of time to train. GPUs are able to speed things up due to their large amount of cores, allowing for parallelism. This is one of the reasons deep learning has become so popular in recent times. Furthermore, with libraries like TensorFlow and PyTorch, deep learning has become a lot easier. TensorFlow and PyTorch are both machine learning libraries in Python that enable users to program and train deep learning models and deploy them to production [25, 26]. That being said, deep learning models tend to be a lot more parameterised than regular machine learning models, which increases space required, computational expensiveness, and prediction time [16].

3.6 Artificial Neural Networks and the Multi-Layer Perceptron

Artificial Neural Networks (ANN) are a type of structure that are heavily based on the way the brain operates. They are a set of connected points called nodes or neurons [23]. They recognise patterns in data from looking at examples of data from the dataset it's trying to identify the pattern from [28]. A Multi-layer Perceptron (MLP) is a basic and traditional form of an ANN. MLPs are a class of supervised feed-forward neural network, which means every input to a node is the output of a previous node which has undergone some transformation. MLPs take in some input vector, which is multiplied by a weight vector with some bias added to the result. At each neuron the product of this is transformed by some activation function so that non-linear relationships can be accounted for. Common activation functions are tanh, ReLU, and sigmoid. This operation happens at each node in each layer with the output of the previous node is transformed and becomes the input to the next, until the final layer where the final result is outputted. Hereafter, a loss function calculates the loss based on the final output and the target value. This loss function is used for the optimisation technique called back-propagation which is how the model learns. A typical cost function for a classification task is cross-entropy error and for regression mean-squared error is very common. The weights and biases are changed based on the result of the loss function which in turn improves the performance of the network [10, 22]. Weights and biases are also known as a model's parameters. These parameters are optimised in the training process of the models. More parameters gives the model more ability to learn the true pattern in the data if optimised correctly, but also more ability to overfit. When the model is doing a prediction, the input data is transformed by the network's forward pass to reach a prediction. Mathematically, here is how the forward pass of an MLP looks in matrix form:

$$\mathbf{A}_l = \sigma_l \left(\mathbf{W}_l^T \mathbf{A}_{l-1} + \mathbf{b}_l \right), \quad l = 1, 2, \ldots, L$$

Where W is the weight matrix on layer l, $\mathbf{A}_{l-1}$ is a matrix of the previous layer's activations, b is the vector of biases, l is the current layer, and σ_l is the activation function on layer l.

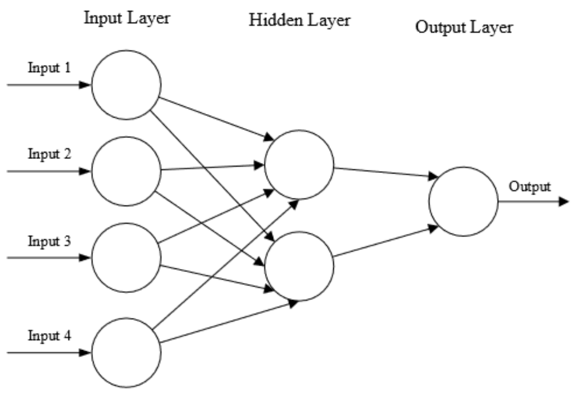

Figure 1: MLP with 4 input nodes,1 hidden layer and one output node [23]

3.7 Convolutional Neural Networks

Convolutional Neural Networks have evolved from the MLP. They are a supervised form of an ANN and they use similar techniques to the MLP for their feed-forward phase and use back-propagation to update their weights and biases [1]. They were originally built for computer vision and image classification and are still popular in the field of computer vision [23]. They are designed to process data with grid-like topology, such as images, and are particularly effective for tasks like image recognition, object detection, and image segmentation. CNNs work by using convolutional layers to scan the input image, detecting local patterns and features, followed by pooling layers that down-sample the data to reduce spatial dimensions [23]. The output is then fed into fully connected layers to produce a final classification or prediction. Through this process, CNNs can learn to recognize complex patterns, enabling them to perform well on a wide range of tasks.

Similar to an MLP, CNNs take in some input vector, transform it to become a single output, and then back-propagate the error to optimize the weights and biases. However, the CNN is more complex than the MLP. The convolutional layer acts as a sliding window over the data, performing a dot product of all the features at the position the window is over. The output of this operation is known as a feature map, which represents the activation of specific features detected by the convolutional filters at different locations in the input records. Each filter in the convolutional layer generates its own feature map, highlighting different aspects of the input data, such as edges, textures, or patterns [13].

The ReLU activation is typically applied to the output for an element-wise activation to introduce non-linearity [23]. The pooling layer then down-samples the result of the activation to reduce parameters. Lastly, the fully connected layer acts in the same way as a regular ANN. This is where the results are transformed using weights, biases, and activation functions to get a final prediction from the CNN [23].

3.7.1 2d Convolutional Neural Networks. 2d-CNNs have a kernel that is 2-dimensional. It is a square window that slides over the data which is able to take information from two dimensions. The kernel size is a hyper-parameter and the larger the kernel size the more information it can take in per position of the kernel. 2d-CNNs perform well in recognising spacial patterns in data due to the two-dimensionality of the kernel. 2d-CNNs tend to have more parameters compared to regular MLPs or 1d-CNNs, often making them slower and more computationally expensive.

3.7.2 1d Convolutional Neural Networks. 1d-CNNs differ from 2d-CNNs by having a kernel that is only one-dimensional. They do not work as well with grid structures but work better when data is in time-series format. 1d-CNNs are especially good at recognising patterns in data that is sequential. Due to the reduced size of the kernel, 1d-CNNs tend to have less parameters than 2d-CNNs, making them more computationally efficient in general [30].

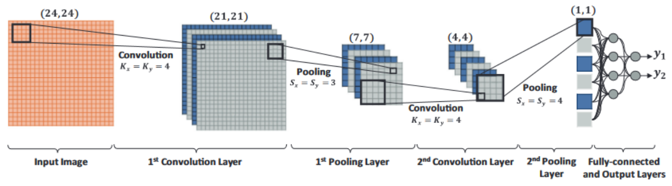

Figure 2: 2d-CNN [18]

3.8 Principal Component Analysis

Principal Component Analysis is a machine learning technique that aims to reduce the dimensionality of a record's feature space. It is an unsupervised type of machine learning since it does not involve any target variable. PCA reduces dimensionality by finding a lower-dimensional representation of a dataset that contains as much of the variation in a dataset as possible. It computes principal components, which are directions in a high dimensional plane that capture the largest amount of variance in the dataset. This is found by using a technique called eigen decomposition. Principal components provide low-dimensional linear surfaces that are closest to the observations. PCA is very commonly used to plot high dimensional datasets. The second or third principal components would be used in order to plot data on a graph [15]. For this project, PCA is used to reduce the amount of parameters in our models. By reducing the input to a model, we reduce the amount of parameters needed which reduces the amount of time it takes to classify a packet and the amount of computational resources our models use. One of the main drawbacks of using PCA compared to other unsupervised techniques, like an auto-encoder, is that it transforms the data linearly [29]. This may result in more information loss for datasets that are non-linear.

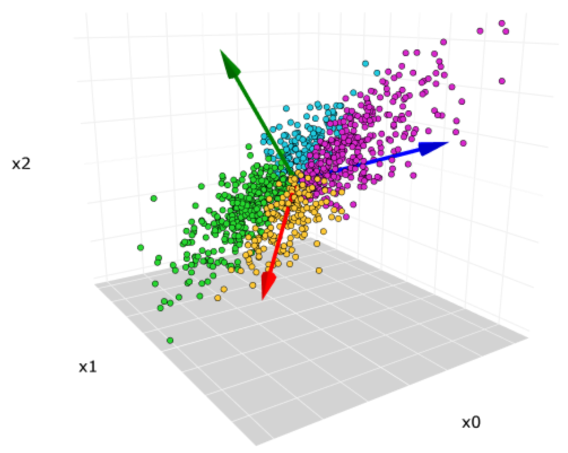

Figure 3: Principal Component Analysis. The first three arrows represent the direction of the first, second and third principal component [34]

4 RELATED WORK

Network traffic classification has had a lot of research conducted on it, especially using deep learning models. Traffic classification is typically done to classify packets into their respective classes (YouTube, Facebook, Teams etc.) or to detect intrusions. Furthermore, packets can be classified based on the contents of their IP payloads or they can be classified by using flows. Flows are aggregations of packets that share the same source port, source IP, destination port, and destination IP. Classifying packets using flows requires a lot of preprocessing before classification, making it less relevant for real time classification. We will be classifying based on IP payloads.

Wang et al. [31] used a flow-based approach for classifying encrypted traffic. They used a 1d-CNN and a 2d-CNN on the ISCXVPN dataset. For the 2d-CNN, they extracted the information from the flows and converted them to a grid format to be fed into the CNN. A 1d vector with the same data was fed into the 1d-CNN. Over a wide range of experiments, the 1d-CNN performed better than the 2d-CNN. Since the data was split into flows, time-series aspects could be extracted and unsurprisingly the 1d-CNN performed better than the 2d-CNN achieving results of 91%. However, the 2d-CNN got 90% which shows the 2d-CNN can perform well on different formats of data and is able to generalise well.

Wang et al. [32] used a 2d-CNN to detect malware over a network and used a series of publicly available datasets. Data was pre-processed into IDX format in order to represent data as a grid before being fed through the model. They achieved accuracy results greater than 90% for each dataset. The datasets were all publicly available and different to ours and the classification type was classifying malware, not traffic type.

Bu et al. [5] used a "large MLP" which consisted of over 4 million parameters in order to classify web data using the ISCXVPN dataset. They achieved an F1 score of 0.974. This shows that MLPs, even though less complex than CNNs, could still be a viable candidate for network traffic classification. However, due to the relatively large parameter count, this could signal that MLPs are only effective with a large amount of parameters.

The work that we will be considering the most is the work by Weisz [33]. Weisz used a "deep" 2d-CNN using a similar dataset that we are using in this project. That model achieved a maximum accuracy of 90%. We will be implementing a similar model to the one Weisz used and use it as a benchmark for the models we produce. This will be used to compare our models over a range of performance metrics. Weisz used a "deep" and "shallow" MLP with 3 and one layer(s) respectively. They achieved results of around 70% accuracy. However, these models were by far the fastest at classification.

From the reviewed literature, it seems that CNN models seem to perform well in terms of accuracy for network traffic classification. 2d-CNNs seem to perform better and are able to generalise better than 1d-CNNs as 1d-CNNs rely more on time-series data. There is little literature that mentions the speed of these models but they tend to have more complex forward passes than MLPs and therefore tend to be slower. From the literature, we observed that MLPs are less commonly used in this traffic classification which could mean they are inferior to CNNs in terms of accuracy generally. This is supported by Weisz's paper. However, MLPs are generally faster than CNNs [8].

5 DESIGN AND IMPLEMENTATION

The methodology used in this project was designed so that the findings can be reproduced. The models are able to run on any type of architecture with the necessary applications and packages installed. Furthermore, the results of this project's experiments are not just to be used for the community network we collected data from, but for any community network or low resource network.

5.1 Dataset

Community network traffic data was used to train and test our deep-learning models. Our research uses datasets collected from the Ocean View community network in Cape Town. The raw data is a collection of raw PCAP files collected at the gateway of the Ocean View community network. The captured data consists of traffic flowing in and out of the network from February 2019 to 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. The data was collected ethically and in this project we will not be showing any sensitive data or data that pertains to anyone individually.

5.2 Preprocessing

The preprocessing stage generated our train, test and validation data from the set of PCAP files. The goal of this stage is to extract the features and a label for each packet from the raw PCAP files. Our target variables are the labels and our input features are the data contained within the IP payload. Packet based classification was used and therefore we want the model to predict what type of packet a packet is without having to split the packets into flows. Information about the packets themselves are contained in the IP payloads, therefore we will be extracting the data from these payloads. Scaling and PCA was then applied, and the data was reshaped for their respective models. The data that is being preprocessed is the data collected from the Ocean View community network.

5.2.1 Packet Labelling. The type of project we are executing is to classify packets into their application type. For example, a packet coming over the router from YouTube should be classified as YouTube packet. Therefore, we need each packet to be labelled because we are performing supervised learning with the label as our target. The model needs to optimise it's weights and biases using the true and predicted labels. Since the data did not come labelled we used a deep packet inspection (DPI) tool called *nDPI* to handle this [9]. This takes the raw PCAP files and outputs a label based on the contents of each file. However, before we fed the data into the models we split them up into flows first in order to make the process smoother and easier for the *nDPI* library. This was done using *pkt2flow* [6], an open source utility program. A study by Bujilow et al. [5] compared nDPI with other packet labellers and concluded that it had a high accuracy and that it was the most reliable of all of the packet labellers tested.

5.2.2 IP Payload Extraction. The next phase of the preprocessing pipeline is to extract the payload information from the IP packets to be used in our model as explanatory variables. Lotfollahi et al. [20] discusses the issue of varying packet lengths. As neural networks require fixed-size inputs, the options are truncating packets to a fixed length or adding zeros to standardise lengths. We will be following a similar method by processing packets into 1480 byte lengths by adding 0s to the ones that are not 1480 bytes long. An open-source Python library called *scapy* processes packets and flows found in PCAP files. Some methods extract IP payloads from packets and convert them to bytes. Thus, obtaining the features needed for each packet. The output of this phase is 1480 features per packet and the label at the beginning of it.

5.2.3 Balancing Data. An important factor in the model's ability to learn patterns is if the data is balanced or not. Unbalanced data leads to biases in the model's prediction, favoring classes that are more common. From the raw PCAPS, the data was very unbalanced with some packets being very over-represented and some very under-represented. Therefore, we decided to balance the dataset by writing a Python script. This was done by filtering out packets with labels that have less than 9000 records and cutting down labels that have over 10000 records down to a 10000. Table 1 is a summary of the dataset:

Packet Label	Number of Packets
Instagram	10000
Windows Update	10000
Facebook	10000
YouTube	10000
WhatsApp	9998
Quic	9995
Gmail	9992
GoogleServices	9935
Microsoft	9789
BitTorrent	9182

Table 1: Distribution of Packets Across Different Labels

5.2.4 Train Test and Validation Splitting. With the data now being in the form of a label and 1480 features, we now need to split them up into train, test and validation sets. The train set is used for the model to learn the pattern and adjust its weights based on the output of the loss function. The validation set is used to evaluate the model's performance on unseen data during training and to indicate if the model is overfitting during training. Lastly, the test set it used to evaluate the final model's performance. We used a 64% - 16% - 20% split for train, validation and test data respectively.

5.2.5 Dimensionality Reduction. Before the input is fed into the model, the dimensions of the input vector are reduced. This is to reduce the amount of data fed into the model per packet which reduces the model's parameter count, increases the models speed to classify packets and reduces the amount of memory used. However, this does come with a reduction in accuracy which will be discussed later in this paper. PCA was used to reduce the input dimensions and the first 100 principal components were used. Before the PCA was applied, the data was standardised using the *StandardScaler* from Scikit-learn.

5.2.6 Reshaping for the CNNs. Since we are using 1d and 2d-CNNs, data cannot be fed into those models like a regular ANN. For the 1d-CNN we added a channel dimension [14] that represents the time-step between packets. This will be the input to the 1d-CNN:

$(n\_samples, \text{components}, 1)$

Since the data is not time-series data the 1 represents the time-step.

For the 2d-CNN the data needs to be reshaped into a grid format. This will be the input to the 2d-CNN:

$$(n\_samples, height, width) \quad \text{s.t.} \quad height \times width = n\_components$$

5.3 Software and Hardware Used

The project was coded in Python and will use TensorFlow as the machine learning library. This library provides models that do not need to be coded from scratch. It provides a framework to train, tune and deploy models [25]. Scikit-learn is another library that we frequently made use of in this project. It gives access to many machine learning tools, such as one-hot encoders, train-test-validation splitters, PCA and more [27]. The models were built using Google Colab, which gave us access to GPUs. Since there are around 100000 records in our dataset, running the models on our own CPUs would take too much time especially for hyper-parameter tuning. Therefore, we used the GPUs from Colab. GPUs made the project more viable because it reduced training time. The experiments were done using an NVIDIA T4 GPU with high RAM provided by Colab.

5.4 Models

The semi-supervised models we created are a 1d-CNN, a 2d-CNN and an MLP. The aim of these models is to find a fast and accurate model that the community network can use confidently. This is why we applied PCA to the dataset to reduce it in order to reduce parameter count of the models. The data fed into the models is the data produced after preprocessing. We will be referring to the models that have had PCA applied to their data as PCA-1d-CNN and PCA-2d-CNN for the 1d and 2d CNNs respectively, and PCA-MLP for the MLP.

5.4.1 Benchmark 2d-CNN. This is the model that was used by Weisz using a similar dataset. This was the "deep 2d-CNN" which had three layers opposed to their "shallow 2d-CNN" which had one. They used tanh for the first layer and ReLU for the second two. Softmax was used as the final activation because this is a classification problem and Adam was used as the optimiser. Categorical cross entropy was used as the loss function.

5.4.2 Semi-Supervised Models. All of the models follow a similar architecture. They all are using the data that has had the PCA applied to it and they all consist of three layers each using ReLU as their activation. Adam was used as the optimiser and L2 regularisation was used with specific regularisation levels used for each model that yielded the best results. Softmax was used as the final activation function and categorical cross entropy was used as the loss function.

5.5 Other Models Considered

One of the shortcomings of PCA is that it transforms data linearly so non-linear patterns in the data could get lost or corrupted. Other unsupervised dimension reduction techniques were tested. Autoencoders are a type of unsupervised deep learning model. They were trained to reduce the input and the hope was that they can capture any non-linear patterns that PCA could not. However, this was not the case and the highest out-of-sample accuracy the models achieved with the data reduced to 100 features was 69%. Furthermore, since they are a type of deep learning model they increased the parameter count significantly for the models which will decrease their packets classified per second. Kernel Principal Analysis (KPCA) was also tested which is a extension of PCA that reduces data in a non-linear way. Using KPCA the data was reduced to 100 features. The data was then fed into the model but the out-of-sample accuracy was averaging at around 21%. This could be due to the fact that the feature space was reduced too much but we were unable to test different numbers of components because KPCA was extremely computationally expensive and required more than an hour to reduce the dataset to 100 components using a GPU. Therefore we decided to use regular PCA. We also considered different numbers of components. 100 was chosen because it yielded results very similar to the models using more components and the less components used, the faster the PCA takes and the less parameters the models have.

Another consideration we had was the use of transformers. Transformers provide state-of-the-art performance various tasks and would likely achieve very good accuracy results. However, transformers are known to be very computationally expensive. This would not work for this project because we are in a resource constrained environment. Furthermore, to train the models it would take a lot of computing power and multiple hours of training on a GPU which would not be viable.

5.6 Hyper-Parameter Tuning

Part of any deep learning project is hyper-parameter tuning. We tuned the hyper-parameters of each model to try to obtain the best accuracies and packets per second classified. We did not perform an automated grid search but, rather performed manual tests with different hyper-parameters. A grid-search would have taken too much time and would have exhausted our training units on Google Colab. Instead, we used a manual search, leveraging our machine learning knowledge and understanding of the model's behavior. We focused on the most promising hyper-parameter combinations and adapted our strategy when we gained new information about models.

The learning rate is an important parameter that determines the model's ability to find the global minimum of the objective function w.r.t the model parameters. We changed the learning rate and ran it on many different configurations. Rates of 0.00005, 0.0001, 0.0005 and 0.001 were tested. All of these learning rates converged to similar results, but the lower ones took more epochs to converge, which is why for all models the learning rate of 0.001 was used, as it took the least epochs to converge.

Secondly, we tried different types of regularization techniques in order to limit overfitting. These include dropout and L2 regularization. We tried dropout levels of 0.05, 0.1 and 0.2 at each level. This resulted in the training process becoming slower and lower accuracy results were observed. L2 regularization was then tested with levels of 0.001, 0.01 and 0.1. All three of these values resulted in the model converging to similar accuracies for the PCA-2d-CNN. Therefore, we decided to choose the model with the L2 term as 0.1, because we want a model that is able to generalise to unseen data more, rather than having a variance that is too high and has fitted to the noise of the training data. In other words, we prefer the model to underfit than overfit. For the PCA-1d-CNN the model with 0.1 as the L2 term did not converge to the same level that the one with 0.001 and 0.01 got. Therefore, we chose 0.01 as the L2 term for this model. Lastly, for the MLP the values used for the CNNs tended to be too high and the model did not converge after 15 epochs, therefore 0.0001 was used.

Different kernel sizes were also tested for the PCA-CNNs. Only sizes of 3 and 5 were tested because for a record of 100 features, 5 was the biggest size usable. Both models saw a parameter count increase when 5 was used although around a 1% increase in accuracy was observed for both models. Therefore, we decided to use a kernel size of 5.

The parameter counts were changed by varying the filter size of each layer. 10 configurations of filter sizes were tested which varied parameter count. Filter sizes ranging from 4 to 1024 were tested. Models with parameter counts of above 1000000 were not considered as these models were not lightweight enough.

6 EXPERIMENTAL METHODOLOGY

In this section, we outline the design, methodology, and rationale behind our experiments aimed at assessing the effectiveness of our models for the classification task for community networks. This evaluation utilizes the train, validation, and test data generated by the preprocessing pipeline we built.

6.1 Experiments

6.1.1 Experiment 1: Parameters vs Accuracy. This experiment was set up to explore how models with a dimension reduced dataset perform compared to the benchmark model in terms of out-of-sample accuracy. Parameter count is used as a proxy for model complexity. However, if there are more parameters there are more operations that need to happen during the forward pass of the model. More parameters should result in less speed, but more accuracy. This experiment involved testing the accuracy of our models while varying the amount of parameters used for each model. We used this formula for test accuracy:

$$\text{Accuracy} = \frac{\text{Number of packets correctly classified}}{\text{Number of packets classified}}$$

For each model we used 10 combinations of filter sizes to increase/reduce parameter count. Each configuration of the model was trained for 15 epochs across all models and we used TensorFlow's *ModelCheckpoint* function to save the model with the highest validation accuracy. This is because model's with higher validation accuracy tend to lead to models with higher out-of-sample accuracy. The experiment was repeated three times for every configuration of the filter sizes. The reason for this is the models produced slightly different results each time they were run. This is due to the initial random guess of the weights and biases in the optimisation process and the stochasticity of the Adam optimiser. Each test accuracy varied by roughly 0.3%-0.5%.

The aim of this experiment is to test the model's accuracy on out-of-sample data (test data). Out-of-sample data is data that the model has not seen before and tells us how well the model can predict after training. It is the ultimate test for model accuracy. Therefore, all of the accuracies presented used out-of-sample data.

6.1.2 Experiment 2: Parameters vs Packets Classified per Second. This experiment was set up to find out if the models using the feature-reduced dataset would outperform the benchmark model. For this experiment we used the same hyper-parameter configurations used in experiment one. We timed how long it would take for the models to evaluate the test data. We then divided length of the test data by the time taken to find out how many packets could be classified per second. We repeated this 10 times for each parameter configuration and the results were averaged. This is because we were observing slightly different results each time we ran the calculation which could be due to the fact that the workload on the GPU will be slightly different at every run.

$$\text{Packets per second} = \frac{\text{Seconds to classify all packets in the test set}}{\text{Number of packets in the test set}}$$

6.1.3 Experiment 3: Peak memory usage and packets classified per second on low resource architectures. The aim of this experiment is to see how different variations of the models perform in low resource environments. We are effectively simulating a community network's environment. We measured the peak memory usage while classifying a single packet for 10 configurations of filter sizes for each model on different CPU architectures. The same packet was used for each experiment. Packets classified per second was also measured on the different CPU architectures. Peak memory was measured instead of average memory usage as peak memory usage is the worst case scenario and is a good test to see if a community network's architecture can handle each model.

For the experiments we will be using two architectures:

(1) Machine 1: Has an 8 core CPU with a clock speed of 1.79 GHz and 16Gb of RAM.
(2) Machine 2: Has a 2 core CPU with a clock speed of 1.10 GHz and 4Gb of RAM.

It is worth noting that when using machine 2, it exhibited frequent freezing episodes, indicating potential memory management issues. However, this could resemble the performance of the machines found in a community network, which is why it was included in this project.

When looking at Figure 4, all of the CNN models have a point where their test accuracies begin to plateau. For the PCA-1d-CNN and the benchmark 2d-CNN this appears to be around the 450000 parameter mark and for the PCA-2d-CNN this seems to be the 600000 parameter mark. The PCA-MLP however, does not really have a clear plateau point, but has a local peak at around the 160000 parameter mark. These points can be seen as the optimal trade-off between accuracy and speed. The models were compared at these parameter counts and the models at their highest parameter counts on machine 1 and 2.

7 RESULTS AND DISCUSSION

In this section the results of the experiments will be discussed.

7.1 Experiment 1

The first experiment was to measure the change in test accuracy as the parameter count increases. The results are in Figure 4:

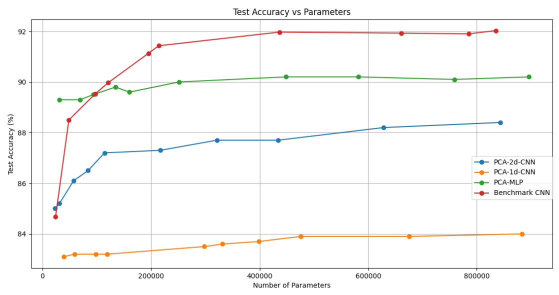

Figure 4: Graph of Test accuracy vs Parameter Count

In Figure 4, we see that all of the model's test accuracies are positively correlated with parameter count which is to be expected as a model with more parameters should be able to learn the overall pattern better as it is more flexible, although more parameters could also lead to overfitting, which could lead to worse out-of-sample accuracy. However, since we applied L2 regularisation, this did not occur.

The benchmark CNN beats the other three models overall in terms of accuracy. This is due to the fact that it has more information at its disposal. 2d-CNNs work well with spatial relationships in data but have shown to perform well on many types of data too and generalise well. When there is more data, the model is able to learn the underlying pattern in the data better because there are more spatial relationships that it can learn from, as there is more data per record. However, the increase in the model's accuracy starts to plateau at around the 450000 parameter count. This could suggest that even though the model has more data to work with than the semi-supervised models, the CNN has reached its capacity. This could be because after a certain point the model has already captured the pattern in the dataset and in each dataset there is some irreducible error, resulting in the model's accuracy plateauing.

The PCA-1d-CNN's accuracy stays fairly consistent when more parameters are added. The gradient of the line is small but positive, indicating that accuracy increases as parameter count increases. However, adding more parameters does not increase the model's performance by much. The model performs very similarly from 450000 to 800000 parameters showing its limit has been reached. This is because the data is not in time-series format. 1d-CNNs are specially suited to time-series data and the data is not in time-series format, which could be the reason that it performs the worst in terms of accuracy. The plateau in accuracy after the 450000 parameter mark suggests that if these models were to be used for real time classification, the model with 450000 should be used. This is because no significant accuracy gains are present and the less parameters, the faster and more lightweight the models become. This demonstrates the ideal trade-off between lightweight models and accuracy.

The PCA-2d-CNN's accuracy seems to be more sensitive to changing parameter counts. The superior accuracy compared to the PCA-1d-CNN is likely due to the fact that the kernel size is 5x5 and is able to capture spatial relationships in the data. However, the accuracy is inferior to that of the benchmark 2d-CNN which is to be expected. PCA does reduce the dataset while keeping as much variance as possible. However, the original dataset will always contain the true pattern in the data which is why the benchmark 2d-CNN with the original dataset performs better on out-of-sample data. With PCA applied, the true pattern can become corrupted, making it more difficult for the model to recognise. From roughly the 20000 to the 800000 parameter mark, the model's accuracy increases by roughly 3%. The model accuracy does exhibit high gradient initially, but it gets lower as more parameters are added. Although, by the last test the accuracy is increasing slightly which could suggest it could perform better if more parameters are added. However, this would no longer result in the model being lightweight. The PCA-2d-CNN performs relatively well, reaching accuracies of above 88% and performs similarly to the benchmark CNN for classification, making it a viable candidate for real time classification.

However, the PCA-MLP outperformed both PCA-CNN models. This could be because the data isn't in time-series format and spatial data is essentially engineered for the 2d-CNNs in the preprocessing stage. The PCA-MLP does not outperform the benchmark 2d-CNN however. The reason is because the MLPs are simpler models. Furthermore, even though the spatial relationships were engineered for the benchmark CNN, it has more data to learn from per record, which suggests 2d-CNNs require a higher amount of features per record to capture the true pattern. From the literature discussed, 2d-CNNs seem to be able to generalise well to different formats of data- data that is not necessarily spatial. However, from looking at our results, this seems to be true but only if there is adequate information in each record. The PCA-2d-CNN struggled with capturing meaningful spatial relationships on records with only 100 features. The PCA-MLP model achieved high accuracies because the dataset is simple and does not require any complex feature extraction. MLPs have shown the ability to perform well on data where the relationship between data is relatively stratight-forward.

To answer research question 1, the PCA-2d-CNN and the PCA-MLP do achieve similar accuracies to the benchmark CNN, with the PCA-MLP showing the best results of over 90% out of the semi-supervised models. However, the benchmark CNN is the most accurate.

7.2 Experiment 2

Figure 5 shows the effect of varying the number of parameters and measuring the packets classified per second for each model.

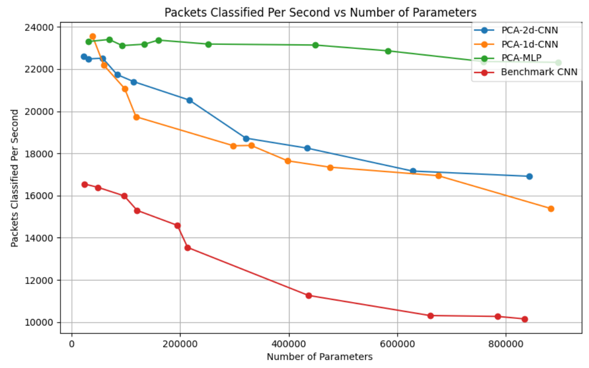

Figure 5: Packets Classified per Second vs Parameter Count

All three curves exhibit negative gradients, which means that when more parameters are added to the models, the model becomes slower to classify packets. This is what we expected because it means that less operations need to happen during the forward pass through the model when there are less parameters resulting in faster classification.

The benchmark 2d-CNN classifies the least packets per second out of the models. This can be attributed to the model's larger feature map at each layer. When a model processes more features per record, it generates larger feature maps at each convolutional layer. Even when the number of parameters is the same as the other models, larger feature maps require more computational resources to process.

The semi-supervised models are a lot faster than the benchmark 2d-CNN. The PCA-CNN models both exhibit similar packets classified per second. This is due to the fact that their feature maps are smaller than the benchmark CNN's. One might expect the PCA-2d-CNN to classify at a rate that is 14.8 times quicker because there are 14.8 times the amount of features in the original dataset. However, there are some overhead costs that do not change, regardless of the size of the dataset, meaning the packets per second does not increase by a factor of 14.8. These overhead costs include the fixed computational overhead required for model initialization, memory management, and data I/O processes that remain constant, regardless of dataset size [19].

However, the PCA-MLP's performance in terms of packets classified per second is far superior to that of the other models. It decreases very slightly as parameters increase. This is due to the simplicity of the MLP architecture compared to the CNN architecture. CNNs have more complex operations than MLPs during the forward pass and have to do more operations when parameters increase, which takes substantially more time than MLP's forward pass. MLPs only do matrix multiplication and apply activation functions during the forward pass.

7.3 Experiment 3

Figure 6 shows the results of measuring peak memory usage with different parameter counts.

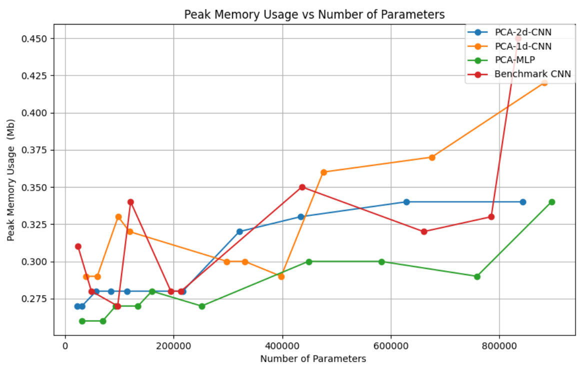

Figure 6: Peak Memory Usage vs Parameter Count

The results of experiment 3 are summarised in tables 2,3,4 and 5.

Model	Params	Packets/s	Params	Packets/s
2d-CNN	436202	1513	834836	1028
PCA-2d-CNN	628170	5927	843274	5244
PCA-1d-CNN	475466	3655	882698	2628
PCA-MLP	160010	19779	896010	11691

Table 2: Packets per second on machine 1 with different parameter counts.

The aim of this experiment is to simulate the low resource environment in a community network. As shown in Figure 5, models with less parameters tend to be able to classify more packets per second than models with more parameters. This proved to be true

Model	Params	Packets/s	Params	Packets/s
2d-CNN	436202	270	834836	225
PCA-2d-CNN	628170	1641	843274	1400
PCA-1d-CNN	475466	774	882698	239
PCA-MLP	160010	13356	896010	10892

Table 3: Packets per second on machine 2 with different parameter counts.

Model	Params	Memory(Mb)	Params	Memory(Mb)
2d-CNN	436202	0.22	834836	0.23
PCA-2d-CNN	628170	0.21	843274	0.22
PCA-1d-CNN	475466	0.23	882698	0.23
PCA-MLP	160010	0.21	896010	0.22

Table 4: Peak memory usage on machine 1 with different parameter counts.

Model	Params	Memory(Mb)	Params	Memory(Mb)
2d-CNN	436202	0.24	834836	0.27
PCA-2d-CNN	628170	0.36	843274	0.33
PCA-1d-CNN	475466	0.4	882698	0.46
PCA-MLP	160010	0.24	896010	0.24

Table 5: Peak memory usage on machine 2 with different parameter counts.

for the tests on the CPUs too. On the CPUs, models with less parameters were able to classify more packets per second than models with more parameters. Furthermore, semi-supervised models were able to classify more packets per second than the benchmark model. All of these results are consistent with the results in Figure 5.

The packets per second for the models on machine 1 and machine 2 were consistent with the results observed in Figure 5, with the PCA-MLP being the quickest and the benchmark CNN being the slowest.

It is also shown that the models that ran on machine 2 consistently were able to classify less packets per second than models ran on machine 1. This is to be expected because machine 2 has less cores, less RAM, and less clock speed than machine 1. Machine 1 was more able to handle the memory requirements of CNNs. CNNs take advantage of parallelism particularly on their convolutional layers. With 2 cores, parallel processing is limited which is why the CNN models are faster on machine 1. The same can be said for the MLP.

The peak memory usages of the models did not have a smooth curve as one might expect. Figure 6 shows the results of testing peak memory usage with different parameter counts to classify a singular packet on each model using a GPU. As expected, more parameters increases peak memory usage. The jaggedness and absence of a smooth curve can be attributed to the memory management system used by the GPU. Some processes get more threads allocated to them at certain run-times than others. The same can be extended to the results seen on the CPUs. The PCA-MLP had the lowest peak memory usage out of all the models on both CPU architectures.

This is due to its simple architecture. The results on machine 1 were stable across all models. All of the models had similar results on machine 1 which could be attributed to the fact that is has more RAM and is able to allocate adequate threads to each process producing consistent results. Surprisingly, on machine 2 the PCA-1d-CNN and the PCA-2d-CNN had a higher peak memory than any of the other models. This was unexpected and can be attributed to the CPU's thread availability at the time of running and the CPU's memory management system which was particularly poor on this machine. Furthermore, it is measuring the peak memory usage and not the average. So this result does not prove the PCA-CNN models are inferior to the other models overall in terms of memory, rather just at their point of highest memory used on this machine. To answer research question 3, the results were not completely definitive. All of the models showed similar peak memory usage on machine 1 but not on machine 2. However, as expected, the PCA-MLP had the lowest peak memory usage on both machines. This is important as the PCA-MLP has emerged as the best in terms of packets per second on both architectures, has very good accuracy and has emerged as the clear candidate for the real-time classification. On machine 1 all of the results are similar with the benchmark CNN narrowly having the highest peak memory usage. Although, on machine 2, the benchmark CNN does not have the highest memory usage and beats the PCA-CNNs.

The model's storage spaces take up between 0.63 Mb and 3.42 Mb, depending on their parameter count and are independent of the architecture. Storage space increases as parameter count increases. The PCA-MLP with 160010 parameters has the lowest storage space, but can reach accuracies of 89% and the benchnark CNN with 834836 parameters has the highest storage space with an accuracy of 92%.

7.4 Conclusions

When considering the resource constrained context of this project, the most suitable candidate for real-time classification seems to be the PCA-MLP. This is simply due to it's speed and relatively high level of accuracy. It outperforms the PCA-CNN models in both accuracy and packets classified per second. One would have to consult the network administrator to find out what model they need in terms of packets per second in order to deem it is a viable candidate for real-time classification. The PCA-MLP uses less memory and still performs well with a relatively low parameter count, which means that if they use the PCA-MLP they can have a model that is not only lightweight in terms of memory use, but also in terms of storage space.

However, a case can be made to use the benchmark 2d-CNN, due to its higher accuracy than the other models although, using this model would not be considering any speed-accuracy trade-off and is likely not to work due to it's poor performance in terms of packets classified per second. One would need to consult the network administrator and ask them if they need a model that is purely accurate, or a model that is less accurate, but fast and lightweight. We suspect that because they are a low resource environment, they need the latter.

The lower parameter version of the models seen in experiment 3 represent the point where the increase in accuracy levels start to drop off as more parameters are added. The reason these model configurations were chosen to be tested on CPUs is because they represent the ideal trade-off between accuracy and speed. Using the lower parameter versions of the models for real time classification would mean sacrificing some accuracy for speed, memory usage and storage space. The community network can look at the results of experiment 3 and decide which model and which version of that model suits them best for their speed needs, and memory and storage constraints.

Something that was not hypothesised was the success of the PCA-MLP and the fact that it beat the PCA-CNN models in both speed and accuracy. This proves the generalisability of the MLP. Furthermore, CNN models clearly flourish when there is more data available. The benchmark 2d-CNN was still the best model in terms of accuracy due to its complexity and abundance of data. The PCA-CNNs did not have the abundance of data the benchmark model had which is why they did not perform as well.

7.5 Limitations

One of the shortcomings of PCA is the fact that it transforms the data linearly. Doing a correlation analysis, we found that roughly 15% of all features on a correlation matrix have an absolute value greater than 0.5 (not double counting). This gives some evidence to suggest that the dataset is moderately linear. The moderate linearity could be why PCA does well to reduce the dataset which is supported by the fact that the PCA-MLP was still able to obtain accuracy levels above 90%. However, more sophisticated feature reduction techniques exist which could capture the elements of the dataset that are non-linear.

Another limitation of this project is the use of the *nDPI* packet labeller. Even though the program compares well to other packet labelling programs, there is still bound to be packets that are labelled incorrectly. Ideally, the labels should be collected during the original collection of the packets.

Time and personal computing resources was also a constraint in this project. Ideally, we would have performed a grid search on the hyper-parameters of the models. However, this is a very time and resource consuming process although, the models were tested extensively on a range of parameters that we thought would yield the right trade-off between speed and accuracy, and the best ones were used.

7.6 Future Work

For future work, we would like to use a different approach to reducing the dimensions of the data in order to capture non-linearity. This could be KPCA with more principal components or another method like t-SNE. We would also like to test the PCA-1d-CNN model again, but this time using time-series data. If we did this, we could also test other models that work well with time-series networks like RNNs and extensions of RNNs.

Transformers can be tested in future work. Although, they do tend to be computationally expensive, we would use one that has been optimised to have a relatively low parameter count and we would evaluate the transformer model using the same experiments as we used in this project.

REFERENCES

[1] Abiodun, O. I., Jantan, A., Omolara, A. E., Dada, K. V., Umar, A. M., Linus, O. U., Arshad, H., Kazaure, A. A., Gana, U., and Kiru, M. U. Comprehensive review of artificial neural network applications to pattern recognition. *IEEE access 7* (2019), 158820–158846.

[2] AlZoman, R. M., and Alenazi, M. J. A comparative study of traffic classification techniques for smart city networks. *Sensors 21*, 14 (2021), 4677.

[3] Azab, A., Khasawneh, M., Alrabaee, S., Choo, K.-K. R., and Sarsour, M. Network traffic classification: Techniques, datasets, and challenges. *Digital Communications and Networks 10*, 3 (2024), 676–692.

[4] Britz, S. Supervised learning, February 2024. Accessed: 2024-08-24.

[5] Bujlow, T., Carela-Español, V., and Barlet-Ros, P. Independent comparison of popular dpi tools for traffic classification. *Computer Networks 76* (2015), 75–89.

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

[7] Chen, Z., He, K., Li, J., and Geng, Y. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *2017 IEEE International conference on big data (big data)* (2017), IEEE, pp. 1271–1276.

[8] Damle, A. Comparing the performance of cnn and mlp in image classification.

[9] Deri, L., Martinelli, M., Bujlow, T., and Cardigliano, A. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)* (2014), IEEE, pp. 617–622.

[10] El Naqa, I., and Murphy, M. J. *What is machine learning?* Springer, 2015.

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

[12] Greene, D., Cunningham, P., and Mayer, R. Unsupervised learning and clustering. *Machine learning techniques for multimedia: Case studies on organization and retrieval* (2008), 51–90.

[13] Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al. Recent advances in convolutional neural networks. *Pattern recognition 77* (2018), 354–377.

[14] Han, D., Yun, S., Heo, B., and Yoo, Y. Rethinking channel dimensions for efficient model design. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition* (2021), pp. 732–741.

[15] Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. *The elements of statistical learning: data mining, inference, and prediction*, vol. 2. Springer, 2009.

[16] Janiesch, C., Zschech, P., and Heinrich, K. Machine learning and deep learning. *Electronic Markets 31*, 3 (2021), 685–695.

[17] Kavanaugh, A., Carroll, J. M., Rosson, M. B., Zin, T. T., and Reese, D. D. Community networks: Where offline communities meet online. *Journal of Computer-Mediated Communication 10*, 4 (2005), JCMC10417.

[18] Kiranyaz, S., Avci, O., Abdeljaber, O., Ince, T., Gabbouj, M., and Inman, D. J. 1d convolutional neural networks and applications: A survey. *Mechanical systems and signal processing 151* (2021), 107398.

[19] Liu, Z., Yang, Q., and Zou, J. Lowering costs and increasing benefits through the ensemble of llms and machine learning models. In *International Conference on Intelligent Computing* (2024), Springer, pp. 368–379.

[20] 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.

[21] Murphy, K. P. *Machine learning: a probabilistic perspective.* MIT press, 2012.

[22] Nielsen, M. A. *Neural networks and deep learning*, vol. 25. Determination press San Francisco, CA, USA, 2015.

[23] O'shea, K., and Nash, R. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458* (2015).

[24] Owens, J. D., Houston, M., Luebke, D., Green, S., Stone, J. E., and Phillips, J. C. Gpu computing. *Proceedings of the IEEE 96*, 5 (2008), 879–899.

[25] Pang, B., Nijkamp, E., and Wu, Y. N. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics 45*, 2 (2020), 227–248.

[26] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems 32* (2019).

[27] 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 Édouard Duchesnay. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research 12*, 85 (2011), 2825–2830.

[28] Ripley, B. D. *Pattern recognition and neural networks.* Cambridge university press, 2007.

[29] Shah, J. H., Sharif, M., Raza, M., and Azeem, A. A survey: Linear and nonlinear pca based face recognition techniques. *Int. Arab J. Inf. Technol. 10*, 6 (2013), 536–545.

[30] Shahid, S. M., Ko, S., and Kwon, S. Performance comparison of 1d and 2d convolutional neural networks for real-time classification of time series sensor data. In *2022 International Conference on Information Networking (ICOIN)* (2022), IEEE, pp. 507–511.

[31] Wang, W., Zhu, M., Wang, J., Zeng, X., and Yang, Z. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE international conference on intelligence and security informatics (ISI)* (2017), IEEE, pp. 43–48.

[32] Wang, W., Zhu, M., Zeng, X., Ye, X., and Sheng, Y. Malware traffic classification using convolutional neural network for representation learning. In *2017 International conference on information networking (ICOIN)* (2017), IEEE, pp. 712–717.

[33] 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.

[34] Yashwanth, S. Pca in ml. *Medium* (2024).

[35] Zhang, J., Chen, X., Xiang, Y., Zhou, W., and Wu, J. Robust network traffic classification. *IEEE/ACM transactions on networking 23*, 4 (2014), 1257–1270.