College of Engineering, Design and Physical Sciences
Department of Electronic and Electrical Engineering
MSc Distributed Computing Systems Engineering

Brunel University London

Big Data Analytics using Deep Learning for Efficient Transmission and Storage

Student ID: 1945959

03/2021

A Dissertation submitted in partial fulfilment of the requirements for the degree of Master of Science

College of Engineering, Design and Physical Sciences

Department of Electronic and Electrical Engineering

MSc Distributed Computing Systems Engineering

Brunel University London

Big Data Analytics using Deep Learning for Efficient Transmission and Storage

Student ID: ______________________

Student's name: ___________________

Signature: _______________________

Declaration: *I have read and I understand the MSc dissertation guidelines on plagiarism and cheating, and I certify that this submission fully complies with these guidelines.*

Abstract

Big data analytics is challenging and costly, not only due to the high volume and complexity of data but also because of the high velocity at which it is generated. Likewise, these properties heavily complicate efficient transmission, storage, and retrieval. Fortunately, deep learning facilitates the handling of such high quantities of data, as it is capable of reducing the dimensionality of the data and extracting expressive features from it. While most recent studies focused on employing these features for specific applications in different areas, this project aims to investigate to what extent the lower dimensionality of the extracted features enables more efficient transmission and storage compared to the raw data.

To that end, multiple autoencoder models were trained in order to extract meaningful features from a dataset. The size of these feature sets was then analysed in relation to their dimensionality. Furthermore, the extracted features were transmitted in a simulated computer network in order to investigate how much traffic can be saved due to their lower dimensionality.

The empirical results show that a lower dimensionality of data increases the information loss, but reduces its size and, consequently, the network traffic induced by its transmission. However, the size of the data also depends on the format in which it is stored. A comparison to PCA indicates that autoencoders achieve a significantly lower information loss, especially for low-dimensional features. Concluding, this work confirms the value of deep learning for big data analytics as it demonstrates that autoencoders can facilitate efficient transmission and storage of data, albeit not without the loss of information.

Contents

List of Figures

List of Tables

1 Introduction

This chapter gives a brief introduction to the topic of this work and points out the fundamental problem it aims to solve. A general solution is proposed and the aim and objectives of the research are defined. Furthermore, an outline of this work is given, briefly summarising the subsequent chapters.

1.1 Background

During the recent years, the amount of data that is generated, transmitted and stored every day has rapidly grown to immense dimensions. By now the term *big data* is on everyone's lips. A lot of companies, research organisations, and other institutions are strongly interested in collecting and analysing such data as it can contain valuable information about various problems such as national intelligence, cyber security or medical informatics [1], just to name a few. Furthermore, analysing productivity-driven data such as log files or metadata gets more and more important [2]. Generally, the volume and complexity of data generated on a daily basis tend to grow exponentially within the next decade [2]. Moreover, this trend is heavily accelerated by the increasing popularity of IoT devices or autonomous driving vehicles that both create a tremendous amount of sensor data that needs to be captured and stored in order to be evaluated [2]. Not to mention social media and other data-intensive applications that further contribute to the rapid growth in data [1], [3].

The immense volume of big data is one of its major positive properties as it contains a lot of valuable information [1]. Having said that, the high volume and complexity of the data, as well as the high velocity at which it is generated, makes an analysis difficult and costly, especially in real-time, because the required computing capacity is vast, which is a fundamental problem in big data analytics [1]. Hence, the collected data often needs to be stored for later processing, which consumes a lot of storage space and furthermore, as most systems nowadays are distributed, implicates the necessity for transmission. Since computing power as well as network and storage capacity are limited, with an increasing amount of data, the significance of the problem also increases.

The solution appears to be simple, at least in theory. To that end, the volume of the data needs to be reduced to facilitate efficient transmission, storage, and retrieval. However, this must be achieved without losing any information that could be of value. Of course, there are countless traditional algorithms that could be used for this, however, they have the disadvantage that they cause high information loss [1]. Fortunately, the field of deep learning offers sophisticated technologies that allow mastering the immense quantities of big data by extracting meaningful features from the data with only little loss of information [1]. To that end, the dimensionality of the data is reduced.

Dimensionality reduction is a common way to lower the complexity of data in order to enable faster and more efficient processing and analysis. Several sources [4], [5], [6] demonstrate that deep neural networks like autoencoders are particularly suitable to extract expressive features from a dataset by reducing its dimensional space. Furthermore, the potential of feature extraction as an efficient approach for information retrieval has already been discussed more than 20 years ago [7]. While most recent researches in this field focus on exploring how the extracted features can be used to store the data in a way that enables a fast and computationally efficient retrieval, for example with semantic hashing [8], [6], [9] and semantic indexing [1] or by using image features to allow for similarity-based image retrieval [10], [6], only little work has been done on investigating if and to what extent the reduced dimensionality of the features can contribute to more efficient storage and transmission. However, this aspect is worth being studied for the reasons mentioned earlier.

To that end, the focus of this work is to examine to what extent a lower dimensionality of data decreases its size and, consequently, the required storage space and the network traffic caused by its transmission, while the actual purpose of use of the features is considered unknown. A special requirement arises from this situation, which is that the features need to describe the original data as precisely as possible because logically the required degree of precision is also unknown then. That is to say, the quality of the features and therefore the information loss must also be considered, as low data size is worthless if the extracted features are unusable due to high loss. The aim and objectives of this work are defined in the following.

1.2 Aim and objectives

This project aims to investigate deep learning for efficient storage and transmission of data. To that end, the size of the data shall be decreased by reducing its dimensionality, but without losing any vital information.

The following objectives were set for the project.

- Carry out a literature review to examine the current state of research and to identify sources that are related to this project.
- Acquire a large and complex dataset as a basis for the experiments.
- Design and train a deep learning model to extract meaningful features from the dataset (with only low information loss).
- Compare the performance of the developed model to a traditional dimensionality reduction method.
- Analyse the size of the extracted features in relation to their dimensionality (with regard to efficient storage).

- Examine the network utilisation caused by the transmission of the extracted features in relation to their dimensionality (for efficient transmission).

1.3 Outline

In this chapter, an introduction to the topic was given and the precise aim and objectives of this work were defined. Chapter 2 summarises and discusses the findings of the literature review and builds a theoretical framework to support the subsequent research. Chapter 3 presents the methods and tools that were utilised. Chapter 4 describes the technical implementation of the research in order to provide an insight into how the experiments were conducted. Furthermore, the obtained results are presented in Chapter 5 and analysed and discussed in Chapter 6. Chapter 7 concludes the research and discusses further work that should or could be done.

2 Literature Review

A literature review was conducted in order to identify a deep learning technology that is most suitable for the demands of this project and to analyse its theoretical strengths and limitations. Moreover, the current state of research was examined and several sources were identified that are related to this project. The findings of the literature review are presented and discussed in this chapter. Furthermore, a theoretical framework is built in order to support the subsequent research.

2.1 Introduction

During the past years, a high quantity of research was conducted to investigate how deep learning could support big data analytics, for example by enabling efficient storage and retrieval of information. Some studies [1], [3] have explored the possible applications and challenges of deep learning in big data analytics in general. Others [4], [5], [11] have examined deep neural networks such as autoencoders for dimensionality reduction and feature extraction. In addition, there are sources that investigate and discuss the applicability of the extracted features for different purposes of use such as similarity-based image retrieval [10], [6], semantic hashing [8], [6], [9], or semantic indexing [1] as a fast and efficient approach to store and retrieve high volumes of information as it is needed for big data analytics. In the following, some important theory is given and the findings of the literature review are presented in more detail.

2.2 Big data analytics

Big data analytics is often described as a central research topic in the field of data science [3], [1]. In order to support the subsequent research, a short introduction to big data analytics is given, explaining its key concepts and highlighting the main challenges that can be addressed with deep learning.

According to Najafabadi et al. [1] the term *big data* refers to "data that exceeds the typical storage, processing, and computing capacity of conventional databases and data analysis techniques" [1] and therefore big data analytics deals with the challenging problem of analysing such vast quantities of data, requiring special methods or tools to allow for efficient processing and storage. They further describe the efficient storage and retrieval of big data as a growing problem, especially due to the high volume and variety of the collected data, but also because of the high velocity at which it is generated. Consequentially, as such data is usually stored distributedly, fast and efficient transmission also gets more and more important. In addition, they mention further challenges in big data analytics such as the high dimensionality of data or its quality in general, which often leads to the necessity of data cleansing and other expensive preprocessing tasks [1].

Deep learning can address several problems in big data analytics [3], [1]. Jan et al. [3] describe deep learning as an important tool for big data analytics as it allows "to automatically identify patterns and extract features from complex unsupervised data without involvement of human" [3]. This is as well confirmed by [1] who describe deep learning algorithms as a promising approach for automated feature extraction. Since the extracted features have a lower dimensionality than the original data, they often can be processed very efficiently using just a simple linear model, which is an important benefit for big data analytics [1]. The features generated by deep learning algorithms are more useful than the high-dimensional raw data because they only contain its most important characteristics and can be processed more efficiently [1]. By contrast, traditional machine learning algorithms are not capable of recognising and extracting such complex nonlinear patterns as they are encountered in big data, which often reflects in worse performance [1].

2.3 Dimensionality reduction

Referred to [1], high-dimensional data can make deep learning algorithms extremely computationally expensive as the high dimensionality vastly contributes to the volume of data. For that reason, dimensionality reduction is an important method for big data analytics, because it allows reducing the dimensions of data by extracting abstract representations from it, the so-called *features* [1]. According to [12], dimensionality reduction is a process where data in a high-dimensional space is transformed into a low-dimensional space. In this context, dimensionality refers to the number of attributes or features in the dataset [12].

The necessity of reducing the dimensional space of data is based on a problem that is often referred to as the *curse of dimensionality*, which is explained in the following according to Géron [13]. For low-dimensional data, the probability is low that a random data point contains any values that are close to a boundary value [13]. With an increasing number of dimensions, however, this probability exponentially increases, as does the available space [13]. The consequence is that high-dimensional datasets are often very sparse, which means that the density of data points in the set is insufficient as the points are scattered [13]. This implies that the data points in the training data are far away from each other, and new samples will also be far from the training samples, which results in slow training, increases the risk of overfitting, and leads to overall worse performance than in lower dimensions [13]. A theoretical solution to overcome this problem is to extremely increase the amount of training data in order to achieve a sufficient density of data points and thus a representative training dataset [13]. However, this is not applicable in practice. Hence, dimensionality reduction is widely used to mitigate the curse of dimensionality [13], [12].

According to Wang et al. [5] reducing the dimensional space of a dataset inevitably leads to information loss. They state that, in order to ensure that the low-dimensional data describes the original as precisely as possible, it is crucial to preserve its most important characteristics [5].

There are several techniques that can be used for dimensionality reduction. The traditional are all statistical methods, of which the principal component analysis (PCA) is the most popular. Thus, it is further described in Section 2.4. However, deep learning approaches such as autoencoders seem to surpass the traditional methods in terms of accuracy [4]. This is as well confirmed by [5]. For this reason, autoencoders are further investigated in Section 2.5.

2.4 Principal component analysis

Principal component analysis (PCA) is a linear algorithm for dimensionality reduction. It is briefly explained in the next paragraph, as described in [13].

To decrease the dimensionality of a dataset, PCA first determines the hyperplane with the lowest average distance from all data points in the set. To that end, its axes are identified based on the variance in the dataset. First, the axis that makes up the largest portion of variance is determined. Second, an axis orthogonal to the first is identified that explains the largest amount of remaining variance. Third, an axis orthogonal to both previous axes is determined, and so forth. These axes are also named the principal components of the data and their number equals the dimensionality of the dataset. Assuming the data shall be reduced to $\boldsymbol{d}$ dimensions, then it is projected onto the hyperplane defined by the first $\boldsymbol{d}$ principal components. Logically, this transformation is linear. Since the principal components are ordered by their portion of the total variance which reflects their significance, this approach preserves as much variance as possible when reducing the dimensionality, because the components with the lowest variance are discarded first whereas those with the highest are kept. However, loss of variance implies loss of information [13].

Some mathematical details are given in the following, according to [14]. Consider $\boldsymbol{X}$ as an $\boldsymbol{n} \times \boldsymbol{p}$ matrix representing the dataset, where $\boldsymbol{n}$ is the number of rows (samples) and $\boldsymbol{p}$ is the number of columns (features). When transforming the dataset into a lower dimension, each row vector $\boldsymbol{x_{(i)}}$ of $\boldsymbol{X}$ is mapped to a new vector of principal component scores $\boldsymbol{t_{(i)}} = \boldsymbol{(t_1,\ldots,t_l)_{(i)}}$ by multiplying $\boldsymbol{x_{(i)}}$ with the $\boldsymbol{p}$-dimensional weight vector $\boldsymbol{w_{(k)}} = \boldsymbol{(w_1,\ldots,w_p)_{(k)}}$, that is $\boldsymbol{t_{k(i)}} = \boldsymbol{x_{(i)}} \cdot \boldsymbol{w_{(k)}}$ for $\boldsymbol{i} = \boldsymbol{1,\ldots,n}$ and $\boldsymbol{k} = \boldsymbol{1,\ldots,l}$. In this process, the variables $\boldsymbol{t_1,\ldots,t_l}$ of $\boldsymbol{t}$ must gradually inherit the maximum possible variance from $\boldsymbol{X}$. Equation 2.1 calculates the weight vector with the maximum variance, that is the weight vector of the first principal component. Once $\boldsymbol{w_{(1)}}$ was found, the first principal component of $\boldsymbol{x_{(i)}}$ can be computed as score $\boldsymbol{t_{1(i)}} = \boldsymbol{x_{(i)}} \cdot \boldsymbol{w_{(1)}}$ [14]. For more details and further reading of how to calculate the other components, see [14].

$$\mathrm{w}_{(1)} = \underset{\|\mathrm{w}\|=1}{\arg\max} \left\{ \|\mathrm{Xw}\|^2 \right\} = \underset{\|\mathrm{w}\|=1}{\arg\max} \left\{ \mathrm{w}^T \mathrm{X}^\mathrm{T} \mathrm{Xw} \right\} \tag{2.1}$$

2.5 Autoencoders

Autoencoders are deep neural networks, widely-used for dimensionality reduction, that try to reconstruct the input data in the output layer [15]. Therefore, the input and output layer both have the same dimensional space, that is the same number of neurons, whereas the hidden layer, in order to achieve the desired dimensionality reduction, must have a lower dimension, because the number of neurons in that layer determines the dimensionality of the extracted features [5], [15]. An autoencoder with such a topology is called an *under-complete* autoencoder [15], [13]. By constraining the hidden layer to have a lower dimensionality than the input, the autoencoder gets an important property, which is the inability of creating a perfect reconstruction of the input by design [15]. Instead, the model is forced to prioritise and choose the most relevant aspects of the input data on its own, and drop the unimportant, in order to approximate the perfect solution, which enables it to learn recognising meaningful patterns and features in the data [15], [13].

Figure 2.1 illustrates the functional schema of an autoencoder according to [16] and extended with information from [15]. Referred to [15], an autoencoder may be imagined as a neural network that consists of two parts: an encoder function $\boldsymbol{h} = \boldsymbol{f(x)}$ that takes the input $\boldsymbol{x}$ and generates the code $\boldsymbol{h}$ and a decoder function $\boldsymbol{r} = \boldsymbol{g(h)}$ that produces a reconstruction $\boldsymbol{r}$ of the input from the code $\boldsymbol{h}$. They state that, in order to achieve a satisfying reconstruction accuracy, the encoder needs to understand which features of the data are most important and therefore should be considered and mapped into the code. Therefore, this reduced or compressed representation of the data, also called the *code*, contains its most salient characteristics, also called the *features* [15]. If the generated code is poor, the decoder won't be able to produce a good result [15].

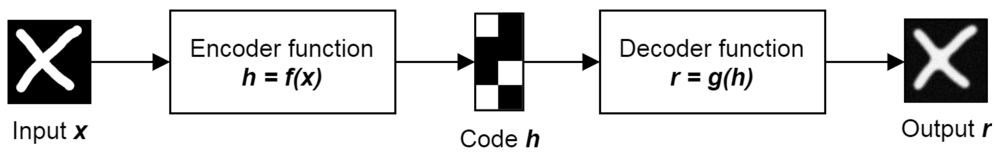


Figure 2.1: Functional schema of an autoencoder (simplified) [16]

Referred to [15], using multiple hidden layers instead of just a single one mostly results in a significantly higher accuracy with the same training dataset due to a better generalisation of the model, as it allows the autoencoder to recognise more complex characteristics of the data. Such an autoencoder, consisting of multiple hidden layers, is referred to as *stacked autoencoder* or *deep autoencoder* [15], [13]. This is as well confirmed by [5] who argue that the useful properties of autoencoders accumulate if they are stacked on top of each other. That is to say, autoencoders profit from depth.

According to [4], the non-linearity of autoencoders is their crucial advantage since it enables the model to learn more complex generalisations in contrast to PCA and other linear methods, which results in a substantially lower loss of information. However, if only linear activation functions are used in the hidden layers, and furthermore the MSE is used as error function, then the accuracy of the autoencoder is equal to the accuracy of PCA [4] since the autoencoder will actually learn to compute the principal components of the data [1], [13].

Autoencoders are trained in an unsupervised manner [5] as they do not require labelled training data, which is a big enabler for them to be used in big data analytics. However, they use supervised learning methods such as backpropagation, and therefore they are sometimes called *self-supervised* [17]. During the training, the backpropagation algorithm adjusts the trainable variables (weights and biases) based on the reconstruction error [15]. The learning process aims at minimising a loss function $\boldsymbol{L(x,g(f(x)))} = \boldsymbol{L(x,r)}$ that compares the input $\boldsymbol{x}$ to the reconstruction $\boldsymbol{r}$ and calculates the difference (error) between both [15].

2.5.1 Performance calculation

The performance of an autoencoder is measured with a metric called *loss* that reflects the amount of lost information as it depicts the dissimilarity between the input and its reconstruction [15]. Thus, it is also called *reconstruction error*. As loss function, it's common to use the mean squared error (MSE) [15]. Since input and output of the autoencoder are both n-dimensional vectors with the same shape, the distance between the two vectors can be easily computed with the MSE as shown in equation 2.2, where the average squared distance between the input $\boldsymbol{X}$ and the output $\boldsymbol{Y}$ is calculated.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(X_i - Y_i)^2 \tag{2.2}$$

2.6 Related research

In the following, the most important literature findings of applications of autoencoders in the field of big data analytics with the emphasis on reducing the dimensionality of data to facilitate fast and efficient storage, transmission and retrieval are presented and analysed.

One of the most popular and trendsetting papers in the field of dimensionality reduction with deep learning was Hinton's and Salakhutdinov's publication [4] in 2006 where they showed that an autoencoder can achieve a lower reconstruction error than PCA. To that end, they trained a deep autoencoder network in which each of the subsequent layers was gradually smaller than the previous layer. The deepest (central) layer of the network only consisted of 30 neurons, which means the data was reduced to only 30 dimensions (or features). For comparison, they conducted a PCA with 30 principal components. They found that the accuracy of the autoencoder was better and it was able to recognise and separate clusters in the data perfectly fine [4]. This paper has notably fuelled interest in deep learning, helping to lay the foundation for big data analytics.

Wang et al. [5] explored autoencoders and their differences to linear and non-linear state-of-the-art dimensionality reduction methods. They also found that autoencoders clearly outperform the traditional algorithms. Moreover, they discovered that autoencoders are not only suitable to reduce the dimensionality of data, but also capable of uncovering repetitive structures and patterns in the data, which is a property they consider useful for various applications. What's more, in their experiments they observed the highest accuracy if the number of neurons in the central layer was close to the intrinsic dimensionality of the dataset. Therefore, they suppose a relation between both [5].

Referred to Meng et al. [11] autoencoders are very good at extracting abstract features from high-dimensional data, but are unable to consider the relations between the features. As a solution, they propose a relation autoencoder model that includes both the features in the data but also the relationships between them. They conclude that the extracted features are more robust if the relationships are considered, which results in a lower reconstruction error [11].

Petscharnig, Lux, and Chatzichristofis [10] examined autoencoders to extract image features for similarity-based retrieval purposes, that is to retrieve images similar to a query image just on the basis of the extracted features. They state that the retrieval performance scales with the dimensionality of the features. Moreover, they conclude that the information loss is higher for a smaller feature set, and therefore the accuracy with which the original data is described is lower, and vice versa [10]. This particular finding is also confirmed by [5].

According to [1], traditional approaches for data storage and retrieval can't keep up with the huge volume and complexity of big data and therefore special solutions are required. One promising approach is semantic hashing, a special technique for storage and retrieval of data that relies on dimensionality reduction and binarisation, which has already has been applied to textual data [8], [9] and to images [6] using autoencoders.

Salakhutdinov and Hinton [8] explored semantic hashing for text data. They found that semantically related data is often placed nearby by many dimensionality reduction algorithms. That is to say, various tasks can be performed much more efficiently on data in a reduced dimensional space as on the original data. In particular, search operations significantly profit from the reduced feature space, because similar data entities are placed nearby. As a consequence, data that is similar to the query data can be found very fast by searching the direct neighbour space [8]. Based on this study, Hinton and Salakhutdinov carried out another research [9] where they applied semantic hashing on entire documents.

Krizhevsky and Hinton [6] applied semantic hashing on image data. To that end, they mapped each image in the dataset to a 28-bit code that kind of represents an extremely reduced feature set where similar images result in similar codes. This allows to rapidly retrieve matching results for a query image as only the 28-bit codes have to be compared. They claim that their approach results in a retrieval time that is absolutely independent of the dataset size. However, since such small codes yield a noticeable loss of information that can result in low accuracy, they also showed that using 256-bit codes can mitigate this effect [6]. Goodfellow et al. [15] complement that such low-dimensional binary codes allow storing the whole dataset in a hash table where each entry is mapped to its corresponding code. When querying data, the search operation only needs to return the entries whose binary codes equal the encoded query, and therefore similar entries can easily be queried by flipping bits in the encoded query [15].

In addition, Najafabadi et al. [1] describe semantic indexing as an essential solution to handle the massive volumes of big data. They propose to use deep learning techniques in order to create abstract representations of the data for semantic indexing rather than using the raw data for this purpose. They claim that this approach can facilitate an efficient information retrieval even for huge datasets as the data is semantically indexed [1].

Géron [13] notes that dimensionality reduction not only decreases the number of features but also the size of the data and therefore could also be used for compression. As an example, he explains that data could be compressed using the PCA algorithm, and decompressed “by applying the inverse transformation of the PCA projection” [13], which reconstructs the original data in a lossy form. This insight could be of value for efficient storage and transmission as the dimensionality is usually reduced anyway to make the data processable. However, he neither gives any details nor explains if this also counts for autoencoders.

Chollet [16] states that autoencoders are not that suitable for data compression as they are highly data-specific, which means that only one particular type of data can be compressed with the encoder, namely the data it was trained with. Furthermore, he claims that it's hard to achieve better results than basic compression algorithms such as JPEG. Having said that, some recent studies [18], [19] have demonstrated the opposite as they actually used autoencoders for image compression. The authors of [18] even claim that their solution can keep up with JPEG 2000. However, both had to utilise very complex and sophisticated neural networks with lots of convolutional layers, yielding an extremely large computational effort compared to traditional compression algorithms. For these reasons, Chollet [16] was eventually proved right. Nevertheless, [18] and [19] show the feasibility and potential of using deep learning for data compression, albeit it becomes obvious that there is still a lot of research necessary.

2.7 Discussion

This work aims to explore deep learning for efficient storage and transmission of big data. The literature review shows that deep learning has been widely investigated in the past years [3], [1] in order to solve the various problems and challenges that arise in big data analytics, caused by the unmanageable high volume of data which heavily complicates efficient transmission, storage and retrieval of information. Several sources [4], [6], [5], [10], [11] demonstrate that deep neural networks such as autoencoders are particularly suitable to master the large volume and complexity of big data, as they are able to extract the most relevant features from it by reducing its dimensionality. Furthermore, their performance is said to excel that of traditional dimensionality reduction algorithms in terms of accuracy [4], [5]. Therefore, autoencoders are a very promising tool for big data analytics and thus are considered as an appropriate deep learning technique for this project.

Several studies have researched autoencoders to facilitate efficient information storage and retrieval, investigating semantic hashing [8], [6], [9] or similarity-based image retrieval [10], [6]. For all of them, however, the main emphasis was to store the data in a way that allows a fast and computationally efficient retrieval, whereas the storage space consumption was not taken into account. Nevertheless, this aspect is important as the available storage space is limited and therefore should be used as efficiently as possible, especially when dealing with such high quantities of data. Logically, the size of the data is decisive in this respect. Moreover, the retrieval of data implies the need for transmission, therefore an efficient transfer is also important, which in turn also depends on the size. Some sources briefly discuss data compression using autoencoders [16], [19], [18] or dimensionality reduction in general [13], indicating that low-dimensional data has a lower size and consequently requires less storage space. Yet none has investigated in-depth to what extent this property can be of value for big data analytics by facilitating a more efficient data storage and transmission, which therefore appears to be a gap in the literature.

2.8 Summary

In this chapter, a theoretical framework was built and related sources were examined and discussed. Eventually, a gap in the literature was identified that shall be addressed in this project. To that end, the focus of this work is to investigate autoencoders with the aim of reducing the size of data and, consequently, the network traffic induced by its transmission in order to allow for storing and transmitting data more efficiently, which seems to be a new application for autoencoders that has not yet been explored before.

3 Methodology

The methods and tools utilised in this project are briefly described and evaluated in this section as they directly affect the quality of the research.

3.1 Choosing the dataset

As a basis for the experiments, a large and complex dataset was searched and identified that meets the criteria of big data. The chosen dataset [20] comes from Kaggle, the world's largest community for data science, and consists of network metadata, more precisely of IP network traffic flows that were collected at the University of Cauca in Colombia in 2017 over a period of six days. The dataset is provided as tabular data in a CSV file and contains almost 3.6 million instances (rows) with 87 attributes or features (columns) such as source and destination IP addresses and ports, protocol, flow duration, transmission rate, and various packet information. Most of the features are numeric, but not all.

Concluding, the dataset is big and complex enough to serve as a sample for big data, knowing that big data usually deals with significantly higher volumes and dimensions. Moreover, the type of data is thematically relevant as an immense volume of network metadata is captured every day in order to be analysed, for example for network anomaly detection. Furthermore, this kind of data appears to not have been investigated before. For these reasons, the dataset is considered suitable for this research.

3.2 Model development

Empirical research was carried out in an iterative manner in order to develop an autoencoder model that is capable of extracting meaningful features from the dataset. That is to say, the performance of the model is decisive as it determines the quality of the features in terms of information loss, and therefore strongly impacts their usability. In each iteration, the parameters of the neural network were adapted and fine-tuned based on the empirical results obtained from the previous iterations. Changes in the topology, as well as varying settings for several training hyperparameters, were investigated. At the end of each iteration, the performance of the trained model was tested and compared to its performance in the prior iterations. The technical details of the development of the autoencoder models are described in Section 4.2.

At some point in the development process, the performance increase between the iterations became very small, which is an indication of being close to the optimal solution. Thus, the training was eventually ended at that point, as the benefit from continuing was too small compared to the additional effort.

3.3 Comparison to PCA

In order to analyse and evaluate the merits of the developed autoencoder model, it is important to compare its performance to another dimensionality reduction technique. Based on the preceding literature review, PCA was chosen for this purpose as it is said to be the most popular method for dimensionality reduction [13] and is often used for comparison [4], [5]. The implementation is described in Section 4.3.

3.4 Prototyping

In order to examine the size of the extracted features in relation to their dimensions, a prototype was implemented that utilises the developed autoencoder model to reduce the dimensionality of the dataset and to store the low-dimensional data into a file. Moreover, the prototype was extended to enable running the model in a computer network in order to investigate if the model can enable the efficient transmission of data by extracting features from a data stream in real-time. Prototyping is a widely-used method to quickly test the suitability of a created piece of software for its intended use. Furthermore, it allows to easily evaluate and compare various potential solutions, which makes it a valuable practice for this research. A detailed description of the developed prototype is given in Section 4.4.

3.5 Computer network simulation

A computer network was simulated in order to investigate to what extent the proposed solution allows for more efficient data transmission. In particular, the reduction of the network traffic in relation to the dimensionality of the extracted features was examined, but also the data rate and transfer duration. To that end, two experiments were conducted, investigating two exemplary computer network scenarios, which are further described in Section 4.5.

Simulating a computer network generally brings advantages over testing in a real network. A simulation provides laboratory conditions without the need of utilising expensive hardware that is difficult to set up. At the same time, it allows producing reliable and reproducible results which can be verified much easier than results obtained in a real network or even on commodity hardware. Furthermore, in a simulation different conditions can be easily set up, for example, the topology of the network can be changed with just a few clicks.

3.6 Tools and software

Different software and tools were utilised to implement the research. The most important ones are briefly described in this section.

3.6.1 TensorFlow

The training of the autoencoder model was implemented with TensorFlow 2.4. TensorFlow [21] is a state-of-the-art technology for machine learning developed by Google. It offers a wide range of functions and an outstanding documentation as well as lots of tutorials. Since version 2.0, TensorFlow supports eager execution which allows evaluating the performance of the model immediately during the training without having to compile it first. Furthermore, TensorFlow provides a visualisation toolkit named TensorBoard that allows visualising summary data captured during the training in a scalar graph. Consequently, important performance metrics can be tracked and analysed in real-time. These qualities make TensorFlow an excellent tool for the highly experimental model development process carried out in this project.

3.6.2 Keras

Keras [22] is an open source library written in Python that allows quick and straightforward implementation of neural networks. It's a high-level API that uses other machine learning frameworks such as TensorFlow as backend and provides easy-to-use methods that hide the complexity of the underlying framework. By now, Keras is already part of the TensorFlow core library. Moreover, it was developed to allow quick and straightforward experimentation, which is an enabler for good and efficient research and thus makes it a very valuable tool for this project.

3.6.3 GPU acceleration

In order to accelerate the training process, TensorFlow was configured to run the calculations on the GPU instead of the CPU, which is especially advantageous when processing large and complex data. Therefore, using the GPU significantly accelerates the training and helps to run the experiments more efficiently, for instance, because a higher number of epochs can be trained in the same time, which increases the chance to find a better solution and therefore is very useful. GPU hardware details are given in Section 3.7.

3.6.4 Scikit-Learn

Scikit-Learn [23] is a software library written in Python that provides a wide range of functions for machine learning-related data processing. In this project, it was used for the data preprocessing as well as to conduct the PCA.

3.6.5 GNS3

GNS3 [24] is a network simulation software that allows to easily build, design and test computer networks in a virtual environment in order to run real-time network simulations. It provides a virtual machine in which Docker images can be set up in order to be embedded and run as virtual hosts in the simulated computer network, which is a very valuable feature for the demands of this project as it enables the execution of arbitrary code on the hosts, such as the developed prototype. An advantage over other network simulation tools is the good documentation and an easy-to-use GUI. Furthermore, GNS3 also includes a wide range of other programmes such as Wireshark, a very powerful tool to capture and analyse the network traffic.

3.7 Computing platform

All experiments were run on Windows 10. The utilised hardware consists of an Intel(R) Core(TM) i5-3350P CPU with 3.10 GHz, 32 GB RAM, a Samsung SSD 750 EVO with 500 GB storage space, and an Nvidia GeForce GTX 1050 Ti GPU with 4 GB GDDR5 RAM that was used with CUDA 11.0.2 and cuDNN 11.0.

3.8 Management of project

Good and careful planning is decisive for the success of each project. In order to ensure a well-organised realisation, an initial time plan was created at the beginning of the project. To that end, the planned work was broken down into smaller tasks that allowed an approximate time estimation. The plan is shown in Figure A.1 in the appendix and covers the whole project life cycle from the beginning of the project to the final submission of the master thesis. However, it doesn't go much into detail since the precise steps for the research were not yet entirely clear at the time of creation.

Eventually, the project plan was not perfectly met due to unforeseen circumstances. To be more precise, the start of the experiments was delayed for a few weeks, mainly due to job-related reasons which resulted in very little time available for this work. Furthermore, the initial plan was not very detailed and some aspects were not even considered. The poster presentation, for instance, was not included since it was not sure whether it will actually take place at all due to the pandemic situation. The final project plan is shown in Figure A.2 in the appendix.

3.9 Summary

In this chapter, the resources, methods and tools that were utilised in this project were described and their value for the successful implementation of the research was explained, as well as their advantages. Furthermore, the management of the project was briefly discussed. While this chapter presented the research methodology, the next chapter describes the actual technical implementation of the research in more detail.

4 Implementation

The implementation of the research is described in this chapter, including the data preprocessing, the training of the autoencoder model with TensorFlow, the prototype development, and furthermore, the computer network simulation with GNS3. All steps that were made to obtain the results presented in Chapter 5 are illustrated in detail.

4.1 Data preprocessing

Depending on its nature, data usually must be prepared before it can be put into a neural network [25]. In general, raw data can only be used very rarely without applying some preprocessing [25]. This also applies to the dataset used in this project. Preprocessing of data is a costly but very important step and should be done carefully as it directly affects the performance of the neural network and therefore the quality of the results.

4.1.1 Conversion of non-numeric values

Neural networks can only process discrete numerical values, therefore all features in the input data need to be of numeric type. However, six of the 87 attributes in the chosen dataset are non-numeric and thus had to be converted first, more precisely the source and destination IP address, timestamp, label, protocol name, and the flow ID.

The IP addresses were transformed into integer values by converting the string representation into a binary number and then into a decimal value. Furthermore, the formatted timestamps were converted into Unix timestamps, and the protocol name and flow label were mapped to integer codes as these are categorical variables. The flow ID, in contrast, is a concatenation of several other attributes (such as IP addresses and ports) and therefore cannot simply be converted into a numeric value. However, since it only consists of data from other columns, it is actually redundant and thus was discarded.

4.1.2 Normalisation

Normalisation of the inputs is a best practice to accelerate the training of the neural network as it leads to faster convergence of the learning curve [25]. In order to equally weigh all features, feature-wise normalisation was applied, which means that the values of each column in the dataset were rescaled separately to be in range 0 to 1.

Feature-wise normalisation is reasonable since the features in the dataset have different scales, for example, the Unix timestamps are very large numbers while network labels consist of only one or two digits. Without normalisation, attributes with larger numbers would have a higher influence on the learning than attributes with small numbers, only caused by their measurement scale but not on their intrinsic nature, making them seem to be more important to the neural network, which is not necessarily correct [25]. Since the relative importance of the attributes is usually unknown, all of them shall be treated as equally important [25]. Normalisation solves these imbalances and therefore is an indispensable step in the preprocessing.

Due to the rescaling, all values in the dataset were converted into floating point numbers. By default, Keras uses 32-bit floating point numbers, which only have a precision of 7 to 8 decimal digits. For use cases such as classification, this degree of precision is most likely sufficient, and it accelerates the training due to faster calculations. In this project, however, the actual use of the data is considered unknown, as is the required degree of precision. Hence, the extracted features shall describe the original data as precisely as possible. That is, it must be ensured that the preprocessing does not already cause some information loss. For this reason, the datatype was explicitly set to 64-bit floating point numbers, which provide a higher precision of 15 to 16 decimal digits.

4.1.3 Shuffling

After the normalisation, the dataset was shuffled, which means that all the samples in the set were randomly interchanged. This ensures that the data points are randomly distributed in the dataset, which logically lowers the variance and therefore helps to reduce the oscillation of the training error. In the chosen dataset, for instance, the samples are ordered in the sequence of capture. This means the timestamp value in the first sample is the smallest whereas the last sample has the largest timestamp, which can be misinterpreted by the model as a certain characteristic in the data, although it isn't. Due to the shuffling, however, this timely order was broken.

4.1.4 Partitioning

In the next step, the data was split into three separate datasets for training, validation, and testing with a ratio of 60% to 20% to 20%. Since the dataset is large, the ratio was not that decisive for the training performance. Even the test partition with a portion of 20% contained over 700,000 samples. The training partition, as the name suggests, was used as input for the training, whereas the validation partition was used to cross-validate the model during the training by checking how it performs on new, unseen data. That is, whether it generalises well or overfits. The test partition, in contrast, was only used after the training to test and evaluate the performance of the trained model on new data.

Now that the data has been prepared, it could eventually be utilised for the training, which is described in the next section.

4.2 Model development

As mentioned in Section 3.2 the autoencoder model development was an extensive experimental process carried out in many iterations. From iteration to iteration, the topology and other hyperparameters were adapted in order to examine their impact on the performance. Changes that had a positive effect were kept or further investigated whereas changes that worsened the performance were immediately discarded. In this way, the model was successively improved until a solution close to the optimum was found. The implementation was done using TensorFlow and Keras. The technical details of this process as well as some background information are given in the following.

4.2.1 Selecting the topology

The topology of a neural network is its most essential characteristic and therefore heavily affects its performance. Which topology is suitable highly depends on the complexity and nature of the data to process. On the one hand, a complex architecture with many layers and lots of neurons gives the model capacity to handle complex data. However, if the model has considerably more capacity than needed, it may start to overfit as it may misinterpret the noise in the training data as certain characteristics. On the other hand, if the capacity of the model is too low, it might not be able to learn from the data which then results in an insufficient performance. For this reason, a suitable topology is best determined in an experimental and iterative manner.

As a starting point for the model development, a shallow topology with only one hidden layer with two neurons was chosen, which was gradually extended with additional layers in the subsequent iterations until the capacity became large enough to enable the model to learn from the data.

After numerous iterations, an appropriate topology was eventually identified from which six candidate models were derived, each having a different number of neurons in the central layer to allow for extracting features with different dimensionality. The models and their topologies are presented in Section 5.1.1.

4.2.2 Training hyperparameters

Besides the topology, various other hyperparameters that were supposed to have an impact on the training performance were examined during the experiments, and thus they are briefly explained in the following.

- **Initialisers**
  TensorFlow offers different presets to initialise the weight matrices of the hidden layers. Each preset has an individual set of initial values that directly affects how fast the model converges. However, which initialiser performs best mainly depends on the nature of the input data.

- **Optimisation algorithm**
  The choice of the optimisation algorithm is substantial for the performance as the optimiser applies the gradients to the trainable variables (weights and biases) and therefore directly affects the learning progress.

- **Activation functions**
  In each neuron, an activation function is applied to calculate its output from the intermediate result. That is, the activation function determines the value range of the output of a neuron [26].

- **Learning rate**
  The learning rate has a direct impact on the training performance as it determines the size of the steps in which the trainable variables are updated. Low values slow down the training whereas high values increase the risk of overshooting the global minimum [15].

- **Momentum coefficient**
  The momentum coefficient allows to drive the optimisation algorithm out of a local minimum and thus can be interesting if the learning gets stuck.

- **Number of epochs**
  The number of epochs determines how often the training data is presented to the neural network, that is how long the network is trained.

4.2.3 Training process

For the training, the datasets were subdivided into batches of consecutive samples, which has various advantages over processing each sample separately. It accelerates the training, not only because it enables parallel processing, but also since the gradients are calculated for the whole batch of samples at once. That is, eventually fewer calculations are necessary which makes it more computationally efficient [27]. Furthermore, it reduces the impact of outliers and variance in the data, which reflects in less oscillation in the learning curve and therefore results in a more stable learning and a more robust convergence [27]. In the end, a batch size of 512 turned out to perform best.

In each training step, one batch of data was processed and the reconstruction error was calculated using the MSE, as described in Section 2.5.1, and recorded to be analysed with TensorBoard. Based on the error, the gradients were calculated and applied using the optimisation algorithm to adjust the trainable variables (weights and biases) of the neural network. Since a lower error means better performance, the training aimed at minimising the loss of the model. On the whole, the model was trained for 10 epochs in each iteration, which is enough to show the trend of the learning curve that was analysed in real-time during the training using TensorBoard. Approaches with very bad performance were stopped and discarded early.

At the end of each iteration, the model was tested after the training to evaluate its performance on new, unseen data by calculating the average reconstruction error on the test partition of the dataset. Unlike the training, the testing was only run for one epoch. While the performance improvement was large during the first iterations, at a later stage of the experiments it became pretty hard to further lower the loss. At this point, the developed model was deemed to be close to the optimum, and thus the development process was ended. The final model, more precisely its topology and the set of hyperparameters it was trained with, was then used as the basis to build the six candidate models, as mentioned before. The resulting models are depicted in Section 5.1. Of course, their performance was also measured, as presented in Section 5.2.

4.2.4 Storing the models

Each of the candidate models was stored for later use. Not only the whole autoencoder but also the encoder was saved separately, because the encoder outputs the features extracted from the input data, and therefore it was of special interest for the subsequent experiments.

4.3 Principal component analysis

As mentioned in Section 3.3 a PCA was conducted on the dataset for different numbers of principal components using Scikit-Learn. After transforming the data into a low-dimensional representation, the extracted features were stored in order to examine their data sizes. Next, the inverse transformation was applied to the extracted features to reconstruct the original data. The error between the original data and its reconstruction was calculated using the same loss function (MSE) as in the autoencoder training. Furthermore, in order to achieve comparable results, the same preprocessing was applied to the data. The performance of the PCA is presented in Section 5.3.

4.4 Prototype development

A prototype was implemented in Python to utilise the developed autoencoder models to extract features from the dataset, and to store them into files. These files were then analysed to explore the relation between the dimensionality of the features and their data size. The findings of this analysis are presented in Section 5.4. Furthermore, the prototype was extended to allow for implementing an online feature extraction in rea-time. Hence, actually two versions of the prototype were developed, an original version and an extended one.

4.4.1 Architecture

The architecture of the prototype is described in the following. Summarising, the prototype utilises a stored autoencoder model to extract features from a given dataset. To be more precise, only the encoder is used. Figure 4.1 shows a schematic illustration of the original prototype, consisting of two main components, the preprocessor and the encoder. After reading the raw data from the source CSV file, the preprocessor transforms it into a format that can be used as input for the encoder by converting all non-numeric values and applying normalisation as described in Section 4.1.1 and Section 4.1.2. In general, the same preprocessing is applied to the data as it was also done during the training. Finally, the encoder converts the preprocessed data into its low-dimensional representation, the features. These features are then stored in the target file, which can either be a CSV or a binary file.

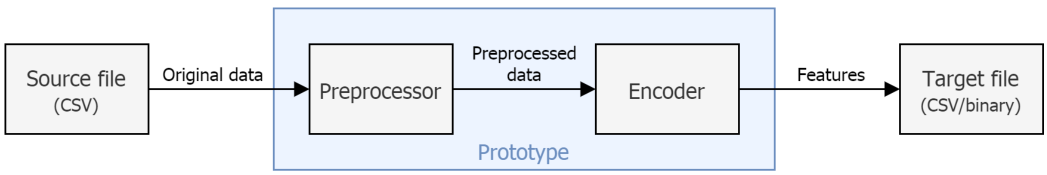


Figure 4.1: Architecture of the original prototype

In order to allow the implementation of an online feature extraction in the simulated computer network, the prototype was extended with two buffers for the input and output, which are implemented as TCP sockets. Instead of reading the data from the source file, it is read from the input buffer. Likewise, instead of writing the data to the target file, it is written to the output buffer. The buffers are required to allow the data to be transmitted and processed simultaneously. For the sake of simplicity, however, the preprocessor and the encoder are only run sequentially. Figure 4.2 illustrates the architecture of the extended prototype.

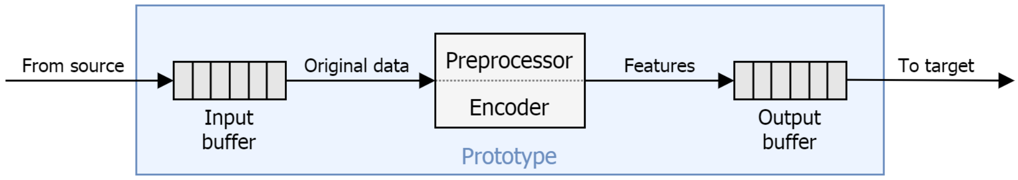


Figure 4.2: Architecture of the extended prototype

4.5 Computer network experiments

In order to explore the impact of the dimensionality of data on the produced network traffic, a computer network was simulated with GNS3 in which two experiments were conducted, investigating two sample applications of the proposed solution. The first was quite basic as only the extracted feature sets were sent from one host to another. The second experiment, in contrast, was more complex as an online feature extraction was implemented as an example application for the proposed solution. In both experiments, the network traffic was captured and analysed with Wireshark. They are described in the following.

4.5.1 Experiment 1: Transmission of the datasets

In the first experiment, the original prototype was used beforehand to extract the features from the test partition of the dataset. Afterwards, these features were transmitted in the network. For comparison, all six candidate models were utilised and therefore six feature sets with different dimensionality and file size were created, stored and transferred.

The simulated computer network was very basic as it consisted of only two hosts connected via a switch. It is illustrated in Figure 4.3. In the experiment, the feature sets, as well as the original data, were sent from the source host to the target and the traffic between the two hosts was captured. The results of this experiment are presented in Section 5.5.1.

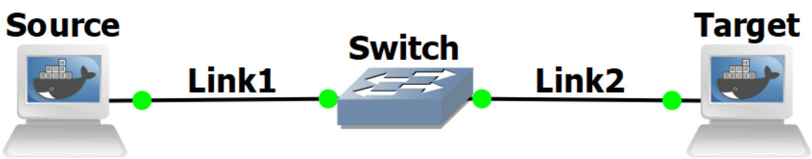


Figure 4.3: Simulated computer network in GNS3 (first experiment)

4.5.2 Experiment 2: Online feature extraction

In the second experiment, the simulated computer network was expanded by a third host that ran the extended prototype in order to extract features from a data stream in real-time, which is called online feature extraction. The test partition of the dataset was sent from the source host to the encoder host as a stream. Meanwhile, on the encoder host, the prototype listened on the input buffer. As soon as 512 samples had been received, they were processed as one batch of data, that is preprocessing was applied and the features were extracted. The output of the encoder was then written into the output buffer and transferred to the target host. The communication was implemented using TCP sockets that were directly connected to the buffers. Since the data was processed in batches of 512 samples, the output of the encoder also comprised 512 low-dimensional data points. As these were written into the output buffer at once, the networking overhead caused by the packet headers could be significantly decreased compared to sending each sample separately to the target host. Figure 4.4 illustrates the simulated computer network. The simulation was run with all six candidate models. Section 5.5.2 summarises the results of this experiment.

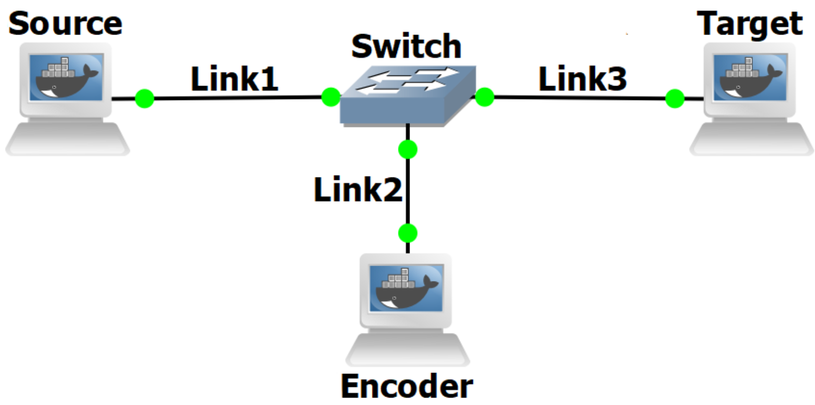


Figure 4.4: Simulated computer network in GNS3 (second experiment)

4.6 Summary

All steps that were conducted during the implementation of the research were described in detail in this chapter. Although the data preprocessing has a direct impact on the training performance, it was just a preparation task and therefore did not provide any relevant results. Likewise, the development of the prototypes alone did not supply any new findings but was necessary for the subsequent experiments. However, the iterative autoencoder training, the performance analysis of both the autoencoder models and PCA, the analysis of the feature set sizes, and the experiments in the simulated computer network led to various findings and results that are presented in the next chapter.

5 Results

In this chapter, the results of the empirical research are illustrated and described in detail, including the topologies and hyperparameters of the created autoencoder models as well as their performance, the PCA results, the sizes of the extracted features in relation to their dimensionality and encoding, and the captured network utilisation of both experiments in the simulated network.

5.1 Autoencoder models

During the model development process, an autoencoder model was designed and trained from which six candidate models were derived. Their topologies and hyperparameters are presented in the following.

5.1.1 Topology

Figure 5.1 illustrates the topologies of the six autoencoder models that resulted from the empirical research that was conducted during the iterative model development process. Each of these models has a different number of neurons in the central layer to enable extracting features with different dimensionality, whereas the input and output layers of all models consist of 86 neurons, which is the number of attributes in the dataset. Furthermore, all models are deep autoencoders, solely consisting of densely connected layers. Moreover, all include four relatively large layers (with 1024, 512, 256, and 128 neurons respectively) in both the encoder and the decoder, providing a lot of capacity that enables the model to recognise certain characteristics in the data, which eventually resulted in a significantly lower reconstruction error. The performance of the models is presented in detail in Section 5.2. However, adding more layers only improved the performance to some extent. That is, further increasing the capacity resulted in an unstable training or led to overfitting.

As advised in [13], all models have a symmetric topology which was found to positively affect the performance, whereas models with asymmetric topology did not perform well. Moreover, it turned out to have a positive effect when the number of neurons in the hidden layers is gradually halved in the encoder and, in turn, gradually doubled in the decoder.

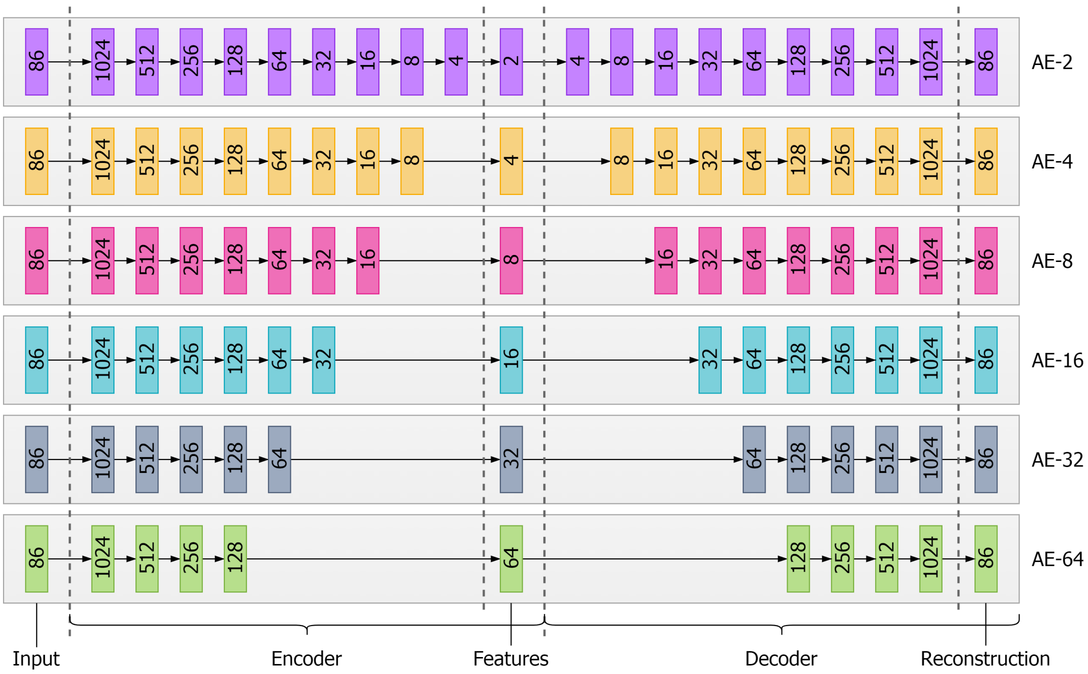


Figure 5.1: Topologies of the different autoencoder models showing the layers and their dimensionality

For the sake of simplicity, the different candidate models were given names and colours which are used for reference in the following sections. For example, AE-2 refers to the autoencoder model with only 2 neurons in the central layer.

What's more, a model with only one single neuron in the central layer was also trained during the experiments. However, the learning curve did not go down but heavily oscillated. In other words, the model was unable to learn and therefore it was not further investigated.

5.1.2 Training parameters

The influence of several training hyperparameters on the performance was investigated during the model development process. The results are presented in the following.

- **Initialisers**
  It turned out that the choice of the initialiser does not affect the global minimum, but only determines how fast the model converges. When training for enough epochs, all initialisers eventually led to the same result. In the experiments, the *He* initialiser with uniform distribution led to the fastest convergence, which is also recommended in [13].

- **Optimisation algorithm**
  The Adam optimiser clearly led to the fastest convergence and thus was used to train the candidate models. It is also recommended in [28].

- **Activation functions**
  Although in the literature [13], [29], [28] ReLU is often recommended for the hidden layers, the hyperbolic tangent function (tanh) performed clearly better, most likely because it can represent negative values which ReLU cannot. In the output layer, however, Sigmoid achieved by far the best performance. This can be explained by the fact that it produces values between 0 and 1, which is exactly the value range of the preprocessed input data and therefore also the range desired in the output.

- **Learning rate**
  The Adam algorithm was found to be very sensitive to a high learning rate, resulting in a heavily oscillating learning curve. In the experiments, a learning rate of 2e-5 turned out to be a good tradeoff between fast convergence and stability of the training.

- **Momentum coefficient**

  Since the backpropagation algorithm was not stuck in a local minimum, the momentum coefficient was not further investigated but left on default.

- **Number of epochs**

  Even after 100 epochs, no overfitting occurred and the learning curves further went down constantly. However, training for 100 epochs took about 15 hours, and therefore the models were not trained any longer.

5.2 Autoencoder performance

As described in Section 2.5.1, the autoencoder performance is measured by calculating the reconstruction error. Figure 5.2 shows a modified TensorBoard screenshot that illustrates the learning curves, more precisely the training error of the candidate models. The x-axis represents the number of steps and epochs during the training, and the y-axis depicts the loss in logarithmic scale.

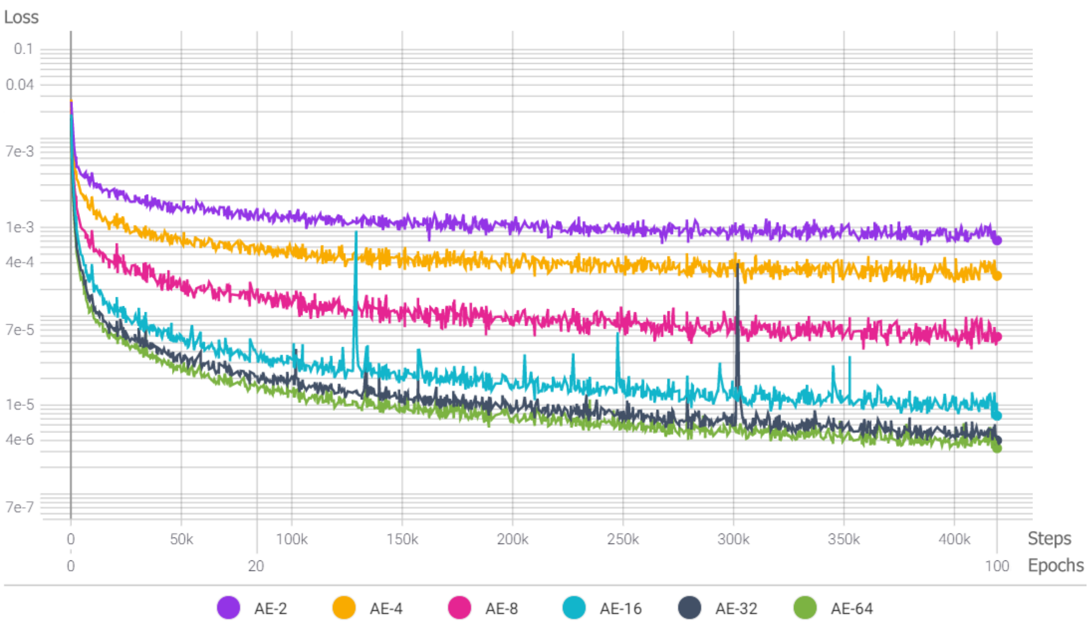


Figure 5.2: Learning curves of the six candidate models during the training

Table 5.1 summarises the training and validation loss of the candidate models at the end of training for 100 epochs. Furthermore, it contains the average loss on the test dataset for each of the models. Since the error slightly oscillated during the training, as visible in Figure 5.2, the training and validation loss is subject to slight fluctuations, and therefore the given values should be treated with caution. The test loss, however, is robust as it is calculated over the whole test dataset, and furthermore, it is more relevant as it reflects the model's performance on entirely new data and therefore expresses the anticipated quality of the extracted features. The results are discussed in Section 6.1.1.

Model	**Training loss**	**Validation loss**	**Test loss**
AE-2 ●	7.1381e-04	9.4547e-04	8.3607e-04
AE-4 ●	2.8593e-04	2.9234e-04	3.2638e-04
AE-8 ●	5.8928e-05	5.9374e-05	6.4497e-05
AE-16 ●	7.5809e-06	8.6956e-06	1.0614e-05
AE-32 ●	4.0060e-06	4.5281e-06	5.0342e-06
AE-64 ●	3.2729e-06	3.7479e-06	3.9950e-06

Table 5.1: Training, validation and test error of the autoencoder models after 100 epochs of training

Based on the loss, the amount of information contained in the extracted features can be calculated with equation 5.1. The result reflects the precision or accuracy with which the original data is described by the features. Considering the test loss in Table 5.1, even the features with only 2 dimensions preserve 99.9164% of the information in the raw data.

$$\text{Accuracy (\%)} = (1 - \text{Loss}) \cdot 100 \tag{5.1}$$

5.3 PCA performance

The results of the PCA are presented in Table 5.2, where the variance and information loss are summarised for different numbers of principal components. It is visible that the first 64 principal components cover 99.99% of the explained variance in the data. Likewise, the last 22 only have a variance of 7.79e-08. The PCA performance is further analysed and discussed in Section 6.1.2 and compared to the performance of the autoencoder models in Section 6.1.3.

Principal components	**Variance**	**Loss**
2	0.5325	1.2797e-02
4	0.7234	7.5722e-03
8	0.8874	3.0812e-03
16	0.9751	6.8098e-04
32	0.9988	3.2273e-05
64	0.9999	2.1328e-09

Table 5.2: Variance and loss for different numbers of principal components

5.4 Dataset sizes

The six autoencoder models were utilised in the original prototype to reduce the dimensionality of the test partition of the dataset. The extracted features were stored in CSV and binary files in order to compare the required storage space with regard to efficient storage. Due to the different number of neurons in the central layers of the candidate models, each of the resulting feature sets had a different dimensionality and therefore a different size. The sizes are summarised in Table 5.3 and discussed in Section 6.2. For comparison, the size of the test partition of the original CSV file is also given as well as the ratio between it and the size of the features. Furthermore, the features extracted with PCA were also stored and analysed, but it turned out that the resulting files have the same sizes. Therefore, the sizes given in Table 5.3 apply to autoencoders as well as to PCA. The format in which the features are stored, however, was found to noticeably impact the required storage space, which is discussed in detail in Section 6.2.

	CSV file		**Binary file**	
Data	**Size (KB)**	**Ratio (%)**	**Size (KB)**	**Ratio (%)**
Original data	371,005	100.00	-	-
64 features ●	1,141,876	307.78	357,905	96.47
32 features ●	570,786	153.85	179,040	48.26
16 features ●	286,534	77.23	89,608	24.15
8 features ●	142,307	38.36	44,891	12.10
4 features ●	71,238	19.20	22,533	6.07
2 features ●	35,823	9.66	11,354	3.06

Table 5.3: Sizes of the extracted features stored as CSV and binary files

5.5 Computer network experiments

In both experiments, the traffic in the computer network was captured and analysed to investigate the efficiency of the transmission. In order to provide reliable results, each experiment was carried out 10 times, and therefore all results presented in the following are the average of the 10 executions. Nevertheless, the results are still subject to slight fluctuations in the low kilobyte or millisecond range respectively. What's more, the extracted features were encoded in binary in both experiments due to the considerably smaller size.

5.5.1 Experiment 1: Transmission of the datasets

In the first experiment, the datasets were simply sent from one host to another. Table 5.4 presents the captured network traffic, more precisely the transmitted packets and bytes in both directions, and the total duration of the transmission. The data rate and the utilised bandwidth are not part of the table as they did not change, because all datasets were transmitted at full speed. The average data rate was about 2.4 MB/s or 19 MBit/s. It is worth noting that about one third of the captured TCP packets were acknowledgements sent from the target host to the source host, and therefore did not carry any payload.

Dataset	Transferred data (KB)	Transferred packets	Duration (seconds)
Original data	397,532	394,473	164.68
64 features ●	383,452	380,430	158.62
32 features ●	191,837	190,351	79.79
16 features ●	95,991	95,234	40.47
8 features ●	48,086	47,541	19.94
4 features ●	24,126	23,978	10.56
2 features ●	12,155	12,114	5.49

Table 5.4: Captured network traffic (first experiment)

5.5.2 Experiment 2: Online feature extraction

In the second experiment, the features were extracted from a data stream in real-time using the extended version of the prototype. Unlike in the first experiment, the utilisation of each link was different because the dimensionality of the data was reduced online. For that reason, the traffic was captured on all three links.

Table 5.5 shows for each of the six candidate models the recorded traffic per link and the total transmission duration, which also includes the run-time of the preprocessor and the encoder. The processing delay per batch, comprising preprocessing and dimensionality reduction, was about 90 milliseconds for all six candidate models, and therefore lower than the transmission delay. That is to say, the data could be processed as fast as it was received.

Encoder	**Link 1 (KB)**	**Link 2 (KB)**	**Link 3 (KB)**	**Duration (seconds)**
AE-64 ●	397,577	783,032	385,455	166.91
AE-32 ●	397,601	590,745	193,114	166.87
AE-16 ●	397,599	495,012	97,412	165.61
AE-8 ●	397,582	447,593	50,011	165.07
AE-4 ●	397,612	422,428	24,817	165.98
AE-2 ●	397,539	410,023	12,485	166.01

Table 5.5: Captured network traffic (second experiment)

Encoder	**Link 1 (MBit/s)**	**Link 3 (MBit/s)**	**Ratio (%)**
AE-64 ●	18.98	18.31	96.47
AE-32 ●	19.08	9.20	48.21
AE-16 ●	18.91	4.57	24.17
AE-8 ●	18.79	2.28	12.13
AE-4 ●	18.89	1.14	6.03
AE-2 ●	19.10	0.58	3.04

Table 5.6: Data rates (second experiment)

Table 5.6 presents for each of the six candidate models the average data rate on link 1 and link 3, which directly reflects the traffic over time. Furthermore, it shows the ratio of the data rate on link 3 in comparison to that on link 1. Logically, the data rate on link 2 coming from the source was the same as on link 1. Likewise, the data rate on link 2 going to the target was equal to that on link 3. For these reasons, only link 1 and 3 are included in the table to avoid redundant information. The results are analysed and discussed in detail in Section 6.3.

5.6 Summary

In this chapter, all results obtained during the research were presented in detail, starting with the created autoencoder models, their topology and performance, and furthermore, the set of training parameters that achieved the lowest information loss. Next, the results of the PCA were given, which are compared to the autoencoders in the next chapter. Moreover, the sizes of the extracted feature sets were depicted in relation to their dimensionality and the format in which they were stored (text or binary). Last but not least, the traffic captured in both experiments in the simulated computer network was presented. In order to examine the importance of the given results, they are analysed and discussed in the next chapter.

6 Discussion

The results presented in Chapter 5 are analysed, evaluated, and discussed in this chapter. To that end, the performance of the developed models is examined and compared to PCA. Furthermore, with regard to efficient storage and transmission of data, the sizes of the feature sets as well as the network simulation results are analysed and discussed. In addition, some limitations are identified.

6.1 Performance evaluation

In the following, the performance of the developed autoencoder models is evaluated and compared to the performance of PCA.

6.1.1 Autoencoder performance

The main purpose of the iterative training process was to develop a model that allows extracting expressive features that contain all vital information from the raw data while at the same time having a significantly smaller size. In this regard, it's obvious that a good performance implies a low information loss.

The model development has demonstrated that the performance is affected by different hyperparameters, from which the topology seems to be the most decisive. Therefore, finding a well-performing combination of parameters is a costly and challenging process that requires a certain level of knowledge and experience. The resulting topologies shown in Section 5.1 confirm the findings from the literature review that autoencoders benefit from depth and symmetry. Furthermore, the results show that increasing the model's capacity can improve its performance, but only to some extent, which indicates that the model needs to fit the complexity of the data. In this respect, the positive impact of symmetry is plausible, as it gives both the encoder and the decoder the same capacity.

The performance of the autoencoder models that emerged from the iterative development process is presented in Section 5.2. The results shown in Figure 5.2 and Table 5.1 indicate that a higher dimensionality in the central layer of the models leads to less information loss. Likewise, a lower dimensionality causes higher loss, as also observed by [10] and [5]. Although this seems logical, it's worth pointing out because it demonstrates that the topology directly affects the performance of the model and furthermore, that the dimensionality of the features is directly related to the accuracy with which the data is described. However, this relation appears to be nonlinear since the performance difference between the models is varying. For instance, the reconstruction error for the models AE-32 and AE-64 is nearly the same whereas the difference between AE-8 and AE-16 is comparatively large. This non-linearity might somewhat depend on the intrinsic dimensionality of the dataset, as also supposed in [5].

When analysing the values in Table 5.1 it becomes obvious that for all models, the difference between training and validation loss is very small, which implies that the models generalised well and no overfitting occurred. This fact coupled with the learning curves shown in Figure 5.2 leads to the assumption that the performance might be further improved when training for even more epochs, as the tendency of the curves suggests that these would also go further down. Despite this, due to time constraints, it was decided to end the training after 100 epochs since that already took more than 15 hours per model.

For AE-2 the difference between training and validation loss is noticeably higher than for the other models, which can be an indication that the model just starts to overfit. However, the test loss lies between the training and validation loss which invalidates this assumption. Likewise, as stated in the results, the learning curves in Figure 5.2 show that the error is subject to slight fluctuations due to noise in the data, which applies to both the training and validation error.

Hence, it is reasonable to assume that the high validation loss is rather caused by a noisy batch of validation data with many outliers. In this regard, it is worth underlining that it's actually possible to reduce the data from 86 to only 2 dimensions with a comparatively low loss, which is not a matter of course.

Although the loss is an important metric to evaluate the models, on its own it's not meaningful enough for the demands of this project. For example, if a model achieves a very low loss but the extracted features are almost as large as the raw data, then it will hardly allow for an efficient storage and transmission. Conversely, features with a very small size are worthless if the loss of information is too high. This issue is further discussed in Section 6.2.

6.1.2 PCA performance

The PCA results given in Section 5.3 demonstrate that for a decreasing number of principal components, the variance also decreases whereas the loss increases. Obviously, the preserved variance depends on the number of principal components, but the results also indicate that there is a close relation between variance and information loss, which is further substantiated by the fact that discarding the last 22 components only causes very low loss since they only cover a fraction of the variance in the data. This relation, however, appears to be nonlinear. For instance, when reducing the number of principal components from 64 to 32, the error significantly increases although only a low amount of variance is lost. In contrast, when using only 2 instead of 4 principal components, a lot of variance is discarded but the difference in the error is comparatively small.

The supposed relation between variance and loss also implies that the number of principal components directly affects the error. Moreover, this implicates that the dimensionality of the features strongly influences the information loss as it is determined by the number of principal components.

6.1.3 Comparison

Figure 6.1 graphically contrasts the loss of PCA and autoencoders, as provided in Table 5.1 and Table 5.2. On the x-axis, the dimensionality of the features is given in descending order, whereas the y-axis represents the reconstruction error in logarithmic scale.

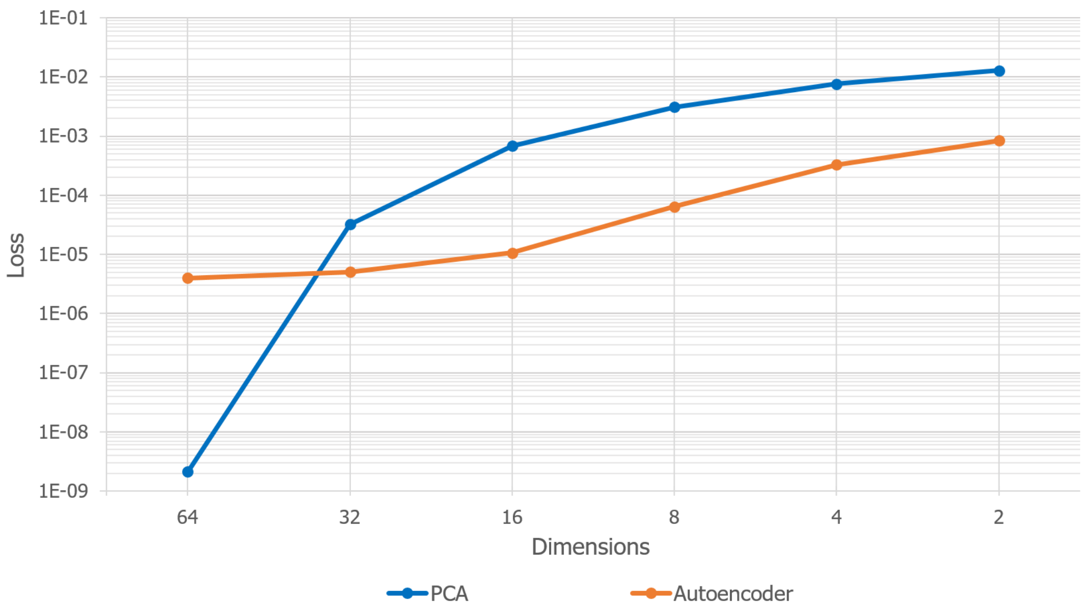


Figure 6.1: Comparison of PCA and autoencoder performance

The analysis of the results in the previous sections has shown that for both techniques the dimensionality of the features directly impacts the loss of information. More precisely, with a decreasing number of dimensions, the loss of information appears to increase. The graphical comparison in Figure 6.1 demonstrates that this relation is nonlinear for both techniques, but also appears to be different for both. Although the loss of PCA with 64 principal components is much lower than the loss of the corresponding autoencoder model, the autoencoders clearly outperform PCA in lower dimensions.

For PCA the loss appears to depend on the ratio of how the variance is distributed over the principal components. When using the first 64 principal components, the last 22 are discarded. Since these only contain a fraction of the variance of the data, the resulting loss is very low. However, the variance distribution highly depends on the nature of the dataset and for different data, the PCA results may look completely different whereas it can be assumed that the autoencoder performance is not sensitive to the variance, which might be one of the reasons why the autoencoders outperform PCA for lower dimensionality.

Furthermore, it is worth mentioning that with longer training or by using a more sophisticated model, the autoencoder loss most likely can be further decreased, whereas the PCA performance is fixed. Having said that, the downside is the time required for the training, which was about 15 hours for 100 epochs per autoencoder model, not to mention the high effort required to find a suitable topology and a well-performing hyperparameter set.

Concluding, the comparison results conform with the findings of [4] and [5]. Furthermore, due to the preceding literature review, it was expected that the autoencoder models outperform PCA, which is confirmed by the obtained results.

6.2 Storage space saving

As briefly explained before, the performance of the autoencoder models is an important metric to evaluate the proposed solution as it determines the quality of the features, but at least equally important for this research is the data size of the extracted features as it determines the consumed storage space.

The results in Table 5.3 indicate that the size of the data is directly related to its dimensionality. Although this finding seems obvious, it is nevertheless worth mentioning as it, in conjunction with the results from the performance evaluation, implies that there is a tradeoff between low data size and low information loss. Moreover, the supposed relation between data size and dimensionality appears to be linear, which suggests that the size of a single feature is independent of the dimensions of the dataset. Consequently, lower dimensionality results in lower storage space consumption, albeit at the expense of information loss.

As described in the results, it appears to make no difference in terms of data size whether the features are extracted with PCA or with one of the autoencoder models, as both techniques yield the same file size, provided that the features have the same dimensionality. However, the quality of the extracted features is different in terms of information loss, which confirms the superiority of the autoencoder models over PCA and therefore supports the adopted approach.

These findings confirm the claim of [13] that PCA can be used for data compression and show that autoencoders can also be utilised for this purpose, which conforms with [18] and [19] who employed autoencoders to compress images. However, decompression (that is the successful reconstruction of the original data) turned out to be a problem for text data, as discussed in Section 6.4.2.

Another important observation regarding the results in Table 5.3 is that the required storage space not only depends on the dimensionality of the features but also on the encoding in which they are stored. Obviously, binary encoding requires significantly less storage space. This can be explained by the fact that the outputs of the encoder are numerical values, more precisely 64-bit floating point numbers. When being stored in binary, such a number consumes exactly 8 bytes. In contrast, when being stored in a CSV file, the number needs to be converted into text. Since a 64-bit floating point number has a precision of 15 to 16 decimal digits, its textual representation in scientific notation requires 24 to 25 ASCII characters, which yields an average size of 24.5 bytes. For these reasons, the size of the extracted features is about three times as large when stored as text. This analysis also confirms the prior assumption that the size per feature is independent of the dimensionality of the data and it also explains why both PCA and autoencoders yield the same file sizes.

Nevertheless, CSV files are a widely used standard to store datasets, but the findings demonstrate that storing the data in binary is much more efficient. Having said that, the downside of binary files is that they are usually application-specific and therefore can only be viewed or processed with the programme they were created with, which is a limitation that often outweighs the benefit of having a smaller data size, and therefore CSV files are still preferred. For this reason, but also because the original dataset was provided as CSV file, it was reasonable to investigate both file formats. When considering the size of the original dataset, it appears to be quite small compared to the feature sets stored as CSV files. This can be explained by the fact that it contains a lot of integer values that only require a low number of ASCII characters to be represented as text.

The prior analysis has shown that the datatype directly affects the data size. As described in Section 4.1.2, Keras was purposely configured to use 64-bit floating point numbers instead of 32-bit numbers due to the higher precision. However, the discussed findings lead to the question of whether this actually makes sense at all, considering that the size of the extracted features could be halved by using 32-bit numbers. Having said that, a datatype with lower precision necessarily increases the loss of information. In the end, this question cannot be answered without further ado, which makes it an open end for further research.

Concluding, lower dimensionality results in lower data size, and therefore storing the extracted features requires less storage space than storing the raw data, provided that the binary format is used. The information loss is the same for both formats, however, binary encoding yields a significantly smaller size. In this regard, it is worth pointing out that even the feature set with only 2 dimensions describes the original data with an accuracy of 99.9164% but only requires 3.06% of its storage space. When storing the extracted features as text, however, the results in Table 5.3 clearly demonstrate that the sets with 64 and 32 dimensions even have a larger size than the original data, although their dimensionality is lower. Therefore, it was decided to only use the binary format for the subsequent experiments as it heavily facilitates efficient storage and transmission in contrast to the text format.

6.3 Network traffic reduction

The network simulation experiments were run in order to examine to what extent the proposed solution allows for an efficient transmission of data by reducing the network traffic. Logically, the produced traffic strongly depends on the size of data to transfer. Therefore, it is reasonable to assume that this also reflects in the simulation results, and hence, the prior discussion of the data size reduction already suggests which findings could be expected in the following analysis. However, these are just assumptions that need to be confirmed.

The results of the first experiment given in Table 5.4 demonstrate that the transmission of the reduced datasets causes less traffic in the computer network than transferring the original dataset. Not only the amount of data but also the number of packets in the network decreases with the dimensionality of the extracted features in a relation that appears to be linear. When comparing the transferred bytes with the dataset sizes summarised in Table 5.3, it is apparent that the captured traffic is around 5% higher than the actual dataset sizes. This can be explained by the networking overhead that comes from the protocol headers of the transmitted packets. Moreover, transferring less data logically implies faster transmission when sending at full speed, which is confirmed by the obtained results. Although these findings seem obvious in some way, they are still worth pointing out as they demonstrate that the proposed solution enables more efficient transmission of data and therefore fulfils the aim of this project. The simulation results therefore also serve as verification.

The results of the second experiment are more interesting as it represents an example application of the proposed solution to process data in real-time, which is highly favourable for big data analytics. Table 5.5 shows the total network traffic captured on the three links. The traffic on link 1 is always the same because the original data is transmitted on that link and its size does not change. For this reason, also the data rate shown in Table 5.6 does not change on link 1. On link 3, in contrast, only the extracted features are transmitted, and therefore the traffic decreases linear with their dimensionality, as does the data rate. Obviously, the utilised model directly affects the amount of traffic in the network as it determines the dimensionality and therefore the size of the data.

What immediately stands out when analysing the results in Table 5.5 is the high traffic on link 2, which is logically caused by the fact that both the original data and the extracted features are transferred on that link. However, it shows that the total traffic in the simulated computer network is higher than in the first experiment, where the data was sent directly from the source to the target, which implies that there is an overhead and the switch has to process more data in total, which is the exact opposite of the desired effect. However, this overhead is mainly caused by the network topology and the way the model is integrated into the computer network and therefore demonstrates that the benefit of the proposed solution also strongly depends on the network conditions.

In summary, without considering the loss, AE-2 appears to be the preferable model as it allows the most efficient data transmission. That is to say, it decreases the traffic on link 3 by almost 97% and only produces about 3% overhead on link 2. The other models also reduce the traffic, but to a lesser extent.

What's more, the total transmission time is practically the same for all utilised models, which indicates that the transmission delay of the original data must be the limiting factor that determines the duration. This, in turn, depends on the size of the original data and the data rate on link 1.

The simulation also demonstrates the feasibility of using deep learning to extract features from a data stream in a computer network in real-time, which is very valuable for big data analytics. However, this is conditional on the system being able to process the data as fast as it is received, which implicates that the processing delay per batch must be lower than the transmission time. Otherwise, the system may become a bottleneck that can negatively impact the whole network. To avoid this, the run time of the utilised model must be as low as possible to be able to handle a high velocity of data. In this regard, the six candidate models hardly differ as their run time mainly depends on their complexity, more precisely on the number of neurons, which is nearly the same for all models.

Concluding, the two experiments demonstrate that the proposed solution can allow for more efficient transmission of data as it reduces its size and therefore the traffic in the computer network. While the first provides some expressive results as it shows that the low-dimensional features can be transferred more efficiently than the raw data, the results of the second experiment largely depend on how the encoder is embedded in the network. Then again, it shows that an online feature extraction is feasible with the proposed solution. After all, both application scenarios just represent example uses of the developed encoders, but there are countless other conceivable use cases.

6.4 Limitations

Although the experiments have generated several convincing results, there are limitations that somewhat affect the expressiveness of the research and the merit of the adopted approach. These limitations are discussed in the following.

6.4.1 Scaling

As described in Section 4.1.2, the data needs to be scaled before being processed by the autoencoder. The problem is that the same scaling needs to be applied to the training data and to new data [25]. While the values in the training data are known and can be used to determine a scaling that fits well for the training dataset, entirely new data points may consist of values outside that range, which is very likely to happen for huge volumes of high-dimensional data. In that case, applying the scaling results in a value outside the range 0 to 1 which is problematic, because the autoencoder can only produce output values inside that range since the Sigmoid activation function is used in the output layer. Thus, it is unable to correctly reconstruct input values outside that range, which limits its usability.

6.4.2 Usability of the data

Although the comparison to PCA has shown that the developed autoencoder models achieve a substantially better performance, it is unknown if the data is still usable for further processing due to the information loss caused by the dimensionality reduction. After all, the performance is just an abstract metric that does not indicate whether the quality of the data is still adequate for different domains, as different use cases require a different degree of precision. However, this question cannot be answered without knowing the actual use of the data and therefore is beyond the scope of this study.

Referred to the usability of the extracted features, an interesting observation to point out is that, although all six candidate models achieve a very low loss, none of them is actually capable of reconstructing the original data from the features in a form that is human-understandable. The problem lies in the nature of the dataset. While images or audio are somewhat resistant to a certain amount of loss, which reduces the level of detail but still allows to recognise the original data, at least in a compressed and lossy form, for text data the loss results in a reconstruction that seems useless. Despite this, the extracted features preserve the most vital characteristics of the original data, not in a form that can be utilised by humans, but in a form that can be understood by other machine learning systems and algorithms. Since tutorials [30], [16] or books [13] often include a demonstration of a successful reconstruction, especially when working on image datasets, it is worth mentioning that this is not applicable for textual data such as the dataset used in this project.

6.4.3 Data-specificity

The chosen dataset is just a sample for big data, but there are countless other types of data. However, according to [16] autoencoders are highly data-specific and therefore can only handle the sort of data they were trained with. This property therefore may constitute a problem for their use in big data analytics as the high variety of big data probably calls for a separate model for each kind of data, whose design and training is associated with a high level of effort as the model development in this project has shown. Furthermore, due to this property, the research cannot tell if autoencoders are generally suitable to handle any sort of data in a similar way as done in this project. Then again, several studies [4], [5], [10] suggest that this is most likely the case, however, there might be very noisy or sparse datasets that even deep learning models can't handle.

6.4.4 Network simulations

A methodological limitation comes from running the computer network simulations with GNS3. Unfortunately, the throughput of the network devices such as switches is strongly limited due to the virtualisation. The highest data rate that could be observed in the experiments was just under 20 MBit/s, which is quite slow. The consequence is that some application scenarios cannot be simulated at all. A further limitation is that the results largely depend on how the model is integrated into the computer network, and therefore they might not be adequate for general conclusions, as discussed before.

6.5 Summary

In this chapter, the results of the experiments were analysed and discussed in detail. The analysis of the autoencoder performance has confirmed the relation between the dimensionality of data and the loss of information. Furthermore, the comparison to PCA has demonstrated that autoencoders provide a better performance, particularly for low dimensions. That is to say, they preserve more vital information in the data than PCA when reducing its dimensionality, which eventually yields a lower reconstruction error. What's more, the analysis of the extracted feature sets has indicated a close relation between data size and dimensionality of the features. However, the encoding in which the data is stored also plays an important role for its size. The analysis of the network traffic captured in the two computer network experiments has shown that the transmission of the low-dimensional features induces less network traffic than transmitting the raw data. Furthermore, the second experiment has demonstrated the feasibility of implementing an online feature extraction with autoencoders. Last but not least, several limitations were identified. Some of them are further discussed in Section 7.2 as an open end for additional research.

7 Conclusions and Future Work

In the last chapter of this work, the whole project is briefly summarised and the conclusions drawn from the research are presented. In addition, some open ends for further work are discussed.

7.1 Conclusions

This project focused on investigating deep learning to support big data analytics by enabling efficient storage and transmission of data. Based on the empirical research carried out in this project, it can be concluded that deep learning can reduce the dimensionality of data with only little information loss. That is to say, it allows for extracting meaningful features from the data that have a significantly smaller size than the original data. Consequently, storing and transmitting these features saves storage space and network traffic compared to the raw data, which enables more efficient storage and transmission.

The literature review has identified autoencoders as a particularly suitable deep learning technique for this research due to their ability to extract meaningful features from data. That is to say, they can reduce its dimensionality with only little loss of information. Low-dimensional data enables faster and more efficient processing and was also supposed to facilitate efficient transmission and storage. However, this was not yet explored before and therefore this work aimed to address this gap in the literature.

Although a comparatively simple dataset was chosen, designing and training a well-performing autoencoder model was costly and time-consuming. Since autoencoders are highly data-specific, a separate model needs to be trained for each sort of data. Coupled with the high variety of big data, this property may become an inhibitor of applying deep learning in big data analytics.

The analysis of the extracted feature sets has indicated that a reduced dimensionality generally yields a lower data size, but has also demonstrated that the encoding of the data noticeably contributes to its size. In this regard, binary encoding turned out to be highly preferable. What's more, the utilised dimensionality reduction method turned out to have no effect on the size of the extracted features, which may raise the question of why using complicated deep learning techniques such as autoencoders at all, when simpler methods like PCA reduce the data size to the same extent. Then again, the performance comparison clearly answers this question as it demonstrates that autoencoders achieve a substantially lower loss than PCA, especially for low-dimensional features. That is, they preserve a higher amount of information in the data, and therefore, although the data size is not different, the quality of the features is, which supports the adopted approach and the use of deep learning in big data analytics.

In line with the experiments in the simulated computer network, an example use of the proposed solution was introduced to allow for more efficient data transmission. Eventually, its suitability has been confirmed as the extracted features not only caused lower network traffic but also could reduce the transmission duration or bandwidth consumption respectively. Furthermore, the feasibility of employing autoencoders to extract features from data in real-time was demonstrated, which can be very helpful to address the various problems in big data analytics. However, the network conditions and the integration of the autoencoder into the computer network directly affect the results, and therefore it's hard to draw general conclusions from that experiment.

Concluding, this work confirms the importance of deep learning in the field of big data analytics as it demonstrates that deep learning can facilitate an efficient transfer and storage of data, which is a highly valuable property to master the massive quantities of big data. The proposed solution allows the extraction of meaningful features that have a significantly smaller size than the original data and thus can be transmitted and stored much more efficiently as they cause less network traffic and require less storage space. However, the lower dimensionality inevitably provokes some information loss. That is to say, it depends on the intended use of the data which degree of precision is required, and therefore which of the six candidate models is most appropriate. Nevertheless, by employing autoencoders for a new purpose of use, this study indicates the manifold application possibilities of deep learning in big data analytics. And although there is still a lot of research necessary in this field, the possibility of efficient storage and transmission of data helps to address several challenges in big data analytics and therefore may even further increase the high interest in deep learning.

7.2 Future work

Although the main research question has successfully been answered, there are still some open questions that may be a good point to build further research on. To that end, some ideas and recommendations for further work are given in the following. Some of them address the limitations discussed in Section 6.4.

The autoencoder models created in this work only consisted of densely-connected layers. Therefore, an obvious approach for further research is to investigate more complex autoencoder architectures to further decrease the loss. While an even better performance won't yield a lower data size, it could further emphasise the superiority of using deep learning in big data analytics. An example to start with is using a convolutional autoencoder, which is often said to perform great on complex datasets [13], or employing a relational autoencoder as introduced by [11], which also considers the relations between the features.

This project solely focused on reducing the dimensionality to enable more efficient storage and transmission of data, while the usability of the extracted features was not investigated, as discussed in Section 6.4.2. Hence, further research could focus on exploring to what extent the loss of information affects the usability of the data when being processed by other algorithms for different use cases. The outcome of these investigations could help to further support the proposed solution or to unveil weaknesses that need to be addressed. Moreover, it could also help to clarify the open question given in the next paragraph.

As already discussed in Section 6.2, storing the extracted features as 32-bit floating point numbers instead of 64-bit numbers could halve the data size and therefore further facilitate efficient storage and transmission. However, using a datatype with lower precision necessarily causes higher loss of information. Hence, an interesting point for further research is to investigate to what extent the error increases when using 32-bit numbers (or even 16-bit numbers), and if that might be preferable over using 64-bit numbers, considering that it allows to further reduce the size of the data.

As mentioned before, the two experiments in the simulated computer network just explored two basic examples of how the system could be used, but there are a lot of other conceivable application scenarios. Hence, another interesting point to further investigate is the potential of optimisation in a more complex computer network, for example where multiple data sources are involved that produce different sorts of data that require different deep learning systems to be processed. Furthermore, problem scenarios could be simulated, for example, where the system running the encoder is too slow and therefore becomes a bottleneck, as already discussed in the previous chapter.

As described in Section 6.4.1, the scaling of data can become a problem. Hence, further research could be carried out in order to overcome this limitation.

References

[1] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald and E. Muharemagic, “Deep learning applications and challenges in big data analytics”, *Journal of Big Data*, vol. 2, 2015.

[2] J. Salomon, *Extracting Value from Data with Deep Learning*, Sep. 2018. [Online]. Available: `https://missinglink.ai/blog/deep-learning/extracting-value-from-data-with-deep-learning` (visited on 10/03/2021).

[3] B. Jan, H. Farman, M. Khan *et al.*, “Deep learning in big data Analytics: A comparative study”, *Computers Electrical Engineering*, vol. 75, pp. 275–287, 2019.

[4] G. E. Hinton and R. Salakhutdinov, “Reducing the Dimensionality of Data with Neural Networks”, *Science*, vol. 313(5786), pp. 504–507, Jul. 2006.

[5] Y. Wang, H. Yao and S. Zhao, “Auto-encoder based dimensionality reduction”, *Neurocomputing*, vol. 184, pp. 232–242, Apr. 2016.

[6] A. Krizhevsky and G. E. Hinton, “Using Very Deep Autoencoders for Content-Based Image Retrieval”, Department of Computer Science, University of Toronto, 2011.

[7] D. Manolescu, *Feature Extraction - A Pattern for Information Retrieval*, 1998.

[8] R. Salakhutdinov and G. Hinton, “Semantic hashing”, *International Journal of Approximate Reasoning*, vol. 50, pp. 969–978, Jul. 2009.

[9] G. E. Hinton and R. Salakhutdinov, “Discovering Binary Codes for Documents by Learning Deep Generative Models”, *Topics in Cognitive Science*, vol. 3 (1), pp. 74–91, 2011.

[10] S. Petscharnig, M. Lux and S. Chatzichristofis, "Dimensionality Reduction for Image Features Using Deep Learning and Autoencoders", ser. CBMI '17, Florence, Italy: Association for Computing Machinery, 2017.

[11] Q. Meng, D. Catchpoole, D. Skillicom and P. J. Kennedy, "Relational autoencoder for feature extraction", *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 364–371, May 2017.

[12] L. van der Maaten, E. Postma and J. van den Herik, "Dimensionality Reduction: A Comparative Review", Oct. 2009.

[13] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition. O'Reilly Media, Inc., 2019.

[14] I. T. Jolliffe, *Principal Component Analysis*. Springer, 2002.

[15] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*. MIT Press, 2016.

[16] F. Chollet, *Building Autoencoders in Keras*, May 2016. [Online]. Available: `https://blog.keras.io/building-autoencoders-in-keras.html` (visited on 11/10/2020).

[17] J. Brownlee, *A Gentle Introduction into LSTM Autoencoders*, Nov. 2018. [Online]. Available: `https://machinelearningmastery.com/lstm-autoencoders` (visited on 12/03/2021).

[18] L. Theis, W. Shi, A. Cunningham and F. Huszár, "Lossy Image Compression with Compressive Autoencoders", 2017.

[19] L. Zhou, C. Cai, Y. Gao, S. Su and J. Wu, "Variational Autoencoder for Low Bit-rate Image Compression", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2018.

[20] J. S. Rojas, *IP Network Traffic Flows Labeled with 75 Apps | Kaggle*, Sep. 2018. [Online]. Available: `https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps` (visited on 14/10/2020).

[21] TensorFlow.org, *TensorFlow Core | Machine Learning for Beginners and Experts*. [Online]. Available: `https://www.tensorflow.org/overview` (visited on 01/01/2021).

[22] Keras.io, *About Keras*. [Online]. Available: `https://keras.io/about` (visited on 01/01/2021).

[23] Scikit-Learn.org, *scikit-learn: machine learning in Python - scikit-learn 0.24.1 documentation*. [Online]. Available: `https://scikit-learn.org/stable` (visited on 06/03/2021).

[24] GNS3.com, *Getting Started with GNS3 | GNS3 Documentation*. [Online]. Available: `https://docs.gns3.com/docs` (visited on 06/03/2021).

[25] Baeldung.com, *Normalizing Inputs of Neural Networks | Baeldung on Computer Science*, Oct. 2020. [Online]. Available: `https://www.baeldung.com/cs/normalizing-inputs-artificial-neural-network` (visited on 28/12/2020).

[26] T. Stöttner, *Why Data should be Normalized before Training a Neural Network*, May 2019. [Online]. Available: `https://towardsdatascience.com/why-data-should-be-normalized-before-training-a-neural-network-c626b7f66c7d` (visited on 08/01/2021).

[27] J. Brownlee, *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*, Jul. 2017. [Online]. Available: `https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size` (visited on 24/03/2021).

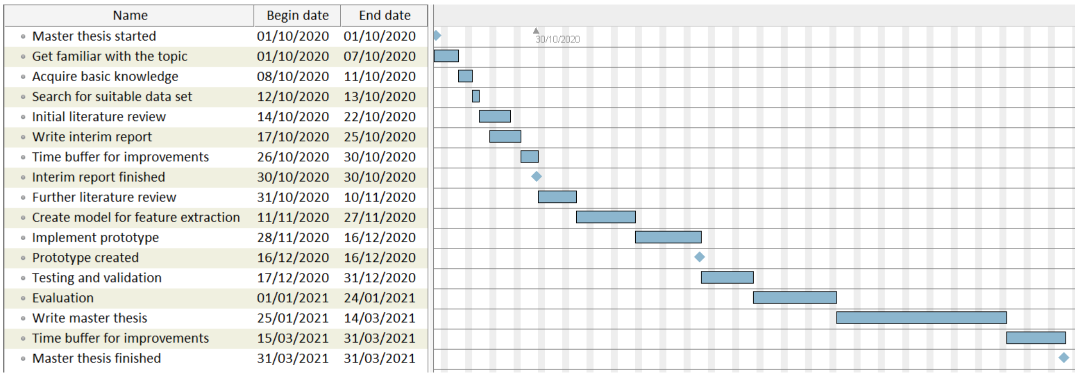
[28] J. J. Moolayil, *Learn Keras for Deep Neural Networks*. Apress, 2019.

[29] F. Chollet, *Deep Learning with Python*. Manning Publications, 2017.

[30] A. F. Agarap, *Implementing an Autoencoder in TensorFlow 2.0*, Mar. 2019. [Online]. Available: `https://afagarap.github.io/2019/03/20/implementing-autoencoder-in-tensorflow-2.0.html` (visited on 27/12/2020).

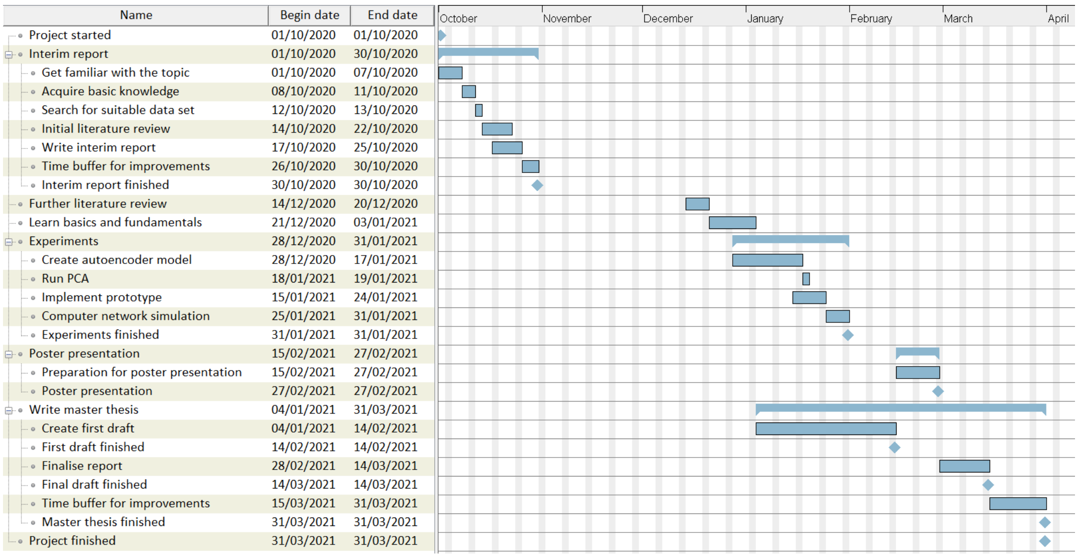
A Appendix

A.1 Project plans



Name	Begin date	End date
Master thesis started	01/10/2020	01/10/2020
Get familiar with the topic	01/10/2020	07/10/2020
Acquire basic knowledge	08/10/2020	11/10/2020
Search for suitable data set	12/10/2020	13/10/2020
Initial literature review	14/10/2020	22/10/2020
Write interim report	17/10/2020	25/10/2020
Time buffer for improvements	26/10/2020	30/10/2020
Interim report finished	30/10/2020	30/10/2020
Further literature review	31/10/2020	10/11/2020
Create model for feature extraction	11/11/2020	27/11/2020
Implement prototype	28/11/2020	16/12/2020
Prototype created	16/12/2020	16/12/2020
Testing and validation	17/12/2020	31/12/2020
Evaluation	01/01/2021	24/01/2021
Write master thesis	25/01/2021	14/03/2021
Time buffer for improvements	15/03/2021	31/03/2021
Master thesis finished	31/03/2021	31/03/2021

Figure A.1: Initial project plan and Gantt chart



Name	Begin date	End date
Project started	01/10/2020	01/10/2020
Interim report	01/10/2020	30/10/2020
Get familiar with the topic	01/10/2020	07/10/2020
Acquire basic knowledge	08/10/2020	11/10/2020
Search for suitable data set	12/10/2020	13/10/2020
Initial literature review	14/10/2020	22/10/2020
Write interim report	17/10/2020	25/10/2020
Time buffer for improvements	26/10/2020	30/10/2020
Interim report finished	30/10/2020	30/10/2020
Further literature review	14/12/2020	20/12/2020
Learn basics and fundamentals	21/12/2020	03/01/2021
Experiments	28/12/2020	31/01/2021
Create autoencoder model	28/12/2020	17/01/2021
Run PCA	18/01/2021	19/01/2021
Implement prototype	15/01/2021	24/01/2021
Computer network simulation	25/01/2021	31/01/2021
Experiments finished	31/01/2021	31/01/2021
Poster presentation	15/02/2021	27/02/2021
Preparation for poster presentation	15/02/2021	27/02/2021
Poster presentation	27/02/2021	27/02/2021
Write master thesis	04/01/2021	31/03/2021
Create first draft	04/01/2021	14/02/2021
First draft finished	14/02/2021	14/02/2021
Finalise report	28/02/2021	14/03/2021
Final draft finished	14/03/2021	14/03/2021
Time buffer for improvements	15/03/2021	31/03/2021
Master thesis finished	31/03/2021	31/03/2021
Project finished	31/03/2021	31/03/2021

Figure A.2: Final project plan and Gantt chart

A.2 Interim report

The interim report is bound at the end of this document and therefore can be found on the following pages.

Big Data Analytics using Deep Learning for Computer Network Optimisation

Distributed Computing Systems Engineering

Interim Report

Student ID: 1945959

Date: 30 October 2020

Brunel University London

School of Engineering and Design

Electronic and Computer Engineering

Contents

List of Figures

1 Introduction

This document was created as part of the master's course Distributed Computing Systems Engineering at Brunel University London. More precisely, it represents an interim report for the master thesis with the topic *Big Data Analytics using Deep Learning for Computer Network Optimisation*. The report shall provide an insight on the scope of the project, including a short introduction into the topic and an initial literature review in order to contextualise the project with the recent advances in the same field of research. Furthermore, it shall define the aim and objectives of the project as well as its deliverables and expected outcomes, but also describe the research methodology that is intended to be utilised. In addition, the report contains a project plan that covers all important steps for a successful project realisation, from the very beginning to the submission of the final master thesis. In that way, this report shall help to identify potential risks at an early stage and set a clearly defined direction of the work. The following section gives a brief introduction to the subject and the problem to be solved.

1.1 Background

During the recent years, the amount of data that is generated, transmitted and stored every day has rapidly grown to immense dimensions. By now the term *Big Data* is on everyone's lips. A lot of companies, research organisations and other institutions are strongly interested in analysing such data in order to create value from it. Big data analytics can be used for commercial purposes like market analysis or future market prediction but also for behavioral analysis of humans or medical research, just to name a few. Traditional data processing approaches quickly reach their limits when it comes to analysing such vast volumes of data in real-time, because they are usually slow and require the data to be well-structured and organised. However, the field of deep learning provides techniques that are capable of handling such giant and unstructured data sets and analysing them in an efficient way.

The main problem is, in order to process the data, it must be captured, transmitted and then stored which requires a fast computer network as well as a lot of storage space. The volume and complexity of the data generated on a daily basis tends to grow exponentially during the next decade. This trend is heavily accelerated by the increasing popularity of IoT devices or autonomous driving vehicles that both create an tremendous amount of sensor data that needs to be captured and stored (in order to be evaluated). At the same time, network bandwidth and storage space are limited. But even if we assume that the performance of the computer networks as well as the available storage space will both increase to a similar extent as the volume of the data, retrieving information from such huge and complex data sets, e.g. by searching for matching objects in the set, will still become more and more challenging, which makes the significance of the problem very clear. [1]

The solution for this problem is basically simple, at least in theory. The volume and complexity of the data must be reduced in order to decrease its size, but without loosing any information that could be of value. Such a reduced data set not only has lower requirements in terms of bandwidth and storage space, but also can be searched much faster in order to find matching objects for a query. Therefore, a requested piece of information can also be retrieved much quicker from a reduced data set then from the original data. Removing redundancy and noise from the data is one possibility to reduce its size, but is not applicable to data which is already compressed.

Deep learning covers a large number of sophisticated technologies in the field of machine learning that are capable of recognising certain characteristics (features) and uncover hidden (cor)relations in unstructured data without requiring human supervision. Several researches (Hinton and Salakhutdinov, 2006; Krizhevsky and Hinton, 2011) [2][3] have shown that deep neural networks like autoencoders are particularly suitable to reduce the dimensional space of a given data set. Dimensionality reduction is a common way to decrease the complexity of data in order to extract features from it by combining the variables in the data and transforming them into a reduced subset of features by applying certain functions. The potential of feature extraction as an efficient approach for information retrieval has already been discussed by Manolescu (1998) [4] more than 20 years ago. However, most recent researches in this field aim at using features for purposes like image or data classification, voice recognition, object or anomaly detection, but rarely for storage and retrieval purposes.

A special requirement that arises from this particular use case is that the created subset of features still needs to describe the original data with a very high accuracy, i.e. without the loss of relevant information. In other words, only if the extracted features are of sufficient quality, they could be used for storage and retrieval purposes instead of the original data.

For that reason, this project focuses on investigating deep learning techniques in order to find out if, how and to what extent they can be actually used as a solution for the described problem. The precise aim and objectives of this work are defined in detail in the next chapter.

2 Aim and objectives

The aim of this project is to develop a solution to search in big data for relevant features that can be used for storage and retrieval purposes, with the intention of optimising the performance of a computer network in terms of storage and bandwidth by using the extracted features rather than the original data.

The following objectives were set for the project.

- Conduct a web search to investigate deep learning techniques and to identify the ones that are most suitable for the demands of this project.
- Carry out a literature review in order to examine the current state of research and to identify approaches that might possibly be related to this project.
- Acquire a suitable data set that can be used as basis for the experimental work. Fortunately, this has already been achieved.
- Develop a model that allows to extract high-quality features from the data set that can afterwards be used for storage and retrieval purposes instead of the original data.
- Implement a prototype that takes the data set as input and uses the created model to extract and store the relevant features from the data.
- Simulate a computer network to test and evaluate the potential of the created solution in terms of optimising the performance of the network regarding storage and bandwidth.

3 Literature review

In line with this interim report, an initial literature search was carried out in order to obtain an overview of the current state of research that is related to this project and the problem to solve. During the last years, a high quantity of researches was conducted in the field of deep learning used for big data analytics. Some studies (Petscharnig, Lux and Chatzichristofis, 2017; Krizhevsky and Hinton, 2011) [5][3] have focused on extracting features from images for classification or retrieval purposes. Others have investigated the potential of autoencoders for dimensionality reduction (Hinton and Salakhutdinov, 2006; Wang, Yao and Zhao, 2016; Meng, Catchpoole, Skillicom and Kennedy, 2017) [2][6][7] in order to extract features from data, or semantic hashing (Salakhutdinov and Hinton, 2009) [8] and semantic indexing (Najafabadi et al., 2015) [9] as a fast and efficient approach to store and retrieve high volumes of information, as it is needed for big data analytics. In the following, the findings of the literature review are discussed and analysed in more detail.

3.1 Dimensionality reduction

Dimensionality reduction is a very important method to handle big data, because it allows to extract features from the data. Data in a high-dimensional space is transformed into a low-dimensional space, i.e. its dimensionality is reduced. In this context, dimensionality refers to the number of attributes or features in the data set. [10]

A very simple example is transforming a table with 100 columns into a table with only 10 columns, while trying to loose as little relevant information or characteristics of the data as possible. However, for more complex data like images or audio this process also is much more complicated.

There are several techniques that can be used for dimensionality reduction. The traditional ones are all statistical approaches, e.g. the Principal Component Analysis (PCA), which is a widely-used statistical method for linear dimensionality reduction that summarises all attributes that describe a similar property to one single attribute. However, such approaches are beyond the scope of this work and are therefore not further investigated. Moreover, deep learning techniques seem to surpass the traditional dimensionality reduction methods in terms of accuracy. [2]

According to Wang et al. (2016) [6] reducing the dimensional space of a data set inevitable leads to information loss. In order to ensure that the low-dimensional data describes the original data as precise as possible, it is crucial to preserve its most important characteristics. [6]

One of the most popular and trendsetting papers in the field of feature extraction using deep learning techniques is Geoffrey Hinton's and Ruslan Salakhutdinov's publication in the Science Magazine in 2006 [2] where they showed that an autoencoder can achieve a lower reconstruction error than the PCA method, i.e. a higher accuracy when reconstructing the original data from the extracted features. In this work, Hinton and Salakhutdinov used a deep autoencoder network in which each of the subsequent layers was gradually smaller than the previous layer. The deepest (central) layer of the network only consisted of 30 neurons, which means the data was reduced to 30 features. For comparison, they conducted a PCA with the same dimensionality, i.e. with 30 principal components. However, the accuracy of the autoencoder was better and furthermore, it turned out to be able to recognise and separate clusters in the data perfectly fine. [2]

Wang et al. (2016) [6] have investigated autoencoders and their differences to state-of-the-art methods (linear and non-linear) for dimensionality reduction. They discovered that autoencoders are not only suitable to reduce the dimensionality of the data, but also can uncover repetitive structures and patterns in the data, which is a property that can be beneficial for several application areas in the field of deep learning. [6][9]

3.2 Autoencoders

Autoencoders are neural networks that try to reconstruct the input data in the output layer. They are widely used for feature extraction purposes. Both, the input and output layer have the same dimensional space, i.e. the same number of neurons [6]. In order to achieve the desired dimensionality reduction, the number of neurons in the hidden layer must be smaller than the number in the input and output layer. In the hidden layer, the input data is transformed into an intermediate representation that should

contain as much vital information of the original data as possible. An autoencoder can have one or multiple hidden layers. Using multiple layers allows the autoencoder to recognise more complex characteristics of the data. The central layer has the lowest number of neurons and therefore acts as a bottleneck where the data compulsorily has the lowest dimensional space. This reduced representation of the data, also called the code, contains its most relevant characteristics, also called the features. Figure 3.1 illustrates the functional schema of an autoencoder. [9][2][6]
According to Wang et al. (2016) [6] the number of neurons in the central layer determines the dimensionality of the features and therefore directly affects the performance. Furthermore, they argue that the useful properties of autoencoders accumulate if multiple autoencoders are stacked on top of each other. [6]

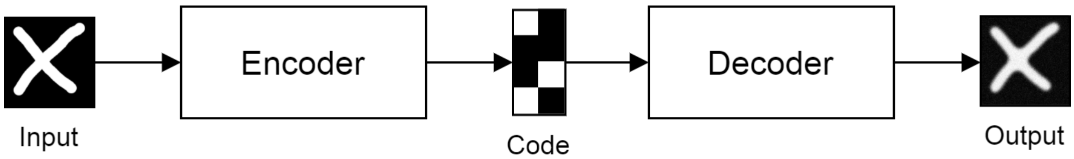


Figure 3.1: Functional schema of an autoencoder (simplified)

Autoencoders are trained in an unsupervised manner [6]. During the training, the back propagation algorithm adjusts the weights based on the reconstruction error that depicts the difference between the input data and the reconstructed data in the output layer. In this way, an autoencoder learns to recognise particular characteristics of the input data which allow to reconstruct the original data from this intermediate representation while discarding irrelevant information. [9][6]

According to Hinton and Salakhutdinov (2006) [2], the non-linearity of autoencoders is their crucial advantage since it enables the model to learn more complex generalisations in contrast to PCA and other linear methods. Instead of only reducing the dimensionality of the data, a non-linear autoencoder is also capable of extracting meaningful representations from it [9]. Due to this fact, the accuracy of autoencoders is noticeably higher, because they can reconstruct the original data from the low-dimensional features with a substantially lower loss of information. However, if using linear activation functions, the accuracy of an autoencoder is very close to the accuracy of a PCA. [2]

Petscharnig, Lux and Chatzichristofis (2017) [5] demonstrated that the accuracy for sparse data sets can be significantly improved when pre-training the neural network model with data from a different but similar domain [5]. The pre-training ensures that the initial weights already are close to a good solution. The power and potential of this approach was also already described by Hinton and Salakhutdinov [2] in 2006.

Another interesting research was conducted by Meng et al. [7] in 2017. They found out that although autoencoders are very good at extracting abstract features from high-dimensional data, they are not capable of considering the relations between the features. As a solution, they proposed a relation autoencoder model that includes both the features in the data but also the relationships between them. They conclude that the extracted features are more robust if the relationships in the data are considered, which results in a lower reconstruction error. [7]
Although Meng et al. (2017) aimed to improve the classification accuracy in their work, their developed solution allows to extract features that describe the original data in a more precise manner than a common autoencoder which does not consider the relations. Therefore, their work should be further investigated for this project.

Petscharnig et al. (2017) [5] have shown that autoencoders can be used to reduce the dimensionality of images in order to extract image features that can be used for similarity-based retrieval purposes, i.e. to retrieve images that are similar to a query image just on the basis of the extracted features. According to their work, the retrieval performance (in terms of speed and accuracy) scales with the dimensionality of the features. The loss of information is higher for a smaller feature set, and therefore the accuracy with which the original data is described is lower, and vice versa. [5]

A similar approach was already investigated before by Krizhevsky and Hinton (2011) [3] who have shown that autoencoders can be used for semantic hashing. In that approach, for each image in the data set, a 28-bit code is calculated and the image is mapped to that code. These codes kind of represent an extremely reduced feature set. Similar images result in similar codes. In this way it is possible to rapidly retrieve matching results for a query image since only the 28-bit codes have to be compared.

Furthermore, Krizhevsky and Hinton (2011) [3] claim that their approach results in a retrieval time that is absolutely independent from the data set size. However, using such small codes leads to a noticeable loss of vital information which can negatively affect the accuracy. Therefore, they also showed that the matching accuracy can be improved by using 256-bit codes [3]. In conclusion, there is a tradeoff between accuracy and retrieval speed which both depend on the code size, i.e. on the dimensionality of the features. [3]

3.3 Semantic indexing

Traditional solutions for data storage and retrieval can't keep up with the huge volume and complexity of big data sets [9]. An interesting finding from Salakhutdinov and Hinton (2008) [8] is that semantically related data is often placed nearby (following the concept of locality) by many techniques for dimensionality reduction. Due to this fact, various tasks can be performed much more efficiently on data in a reduced dimensional space as on the original data. For example classification, or search operations (queries) significantly profit from the reduced feature space, because similar data entities are placed nearby. As a consequence, data which is similar to the query data can be found very fast by searching the neighbour space. [8][9]

Najafabadi et al. (2015) [9] describe semantic indexing as an essential solution to handle the massive volumes of big data. They propose to use deep learning techniques in order to create abstract representations of the data that can be used for semantic indexing rather than using the raw data for this purpose. Their approach can facilitate an efficient information retrieval even for huge data sets, because the data is semantically indexed, and therefore might be of interest for this project. [9]

3.4 Conclusions

In order to find a solution for the problem this project investigates, a way must be found to reduce the dimensionality of the data and extract features that can be used for storage and retrieval purposes. In this regard, the quality of the features is essential as they must preserve the vital information in the data.

The initial literature review has shown that a lot of related researches have been carried out recently. According to several publications (Hinton and Salakhutdinov, 2006; Wang et al., 2016; Petscharnig et al., 2017; Meng et al., 2017) [2][6][5][7] autoencoders are the deep learning technique that is particularly suitable for feature extraction purposes. Some approaches (Krizhevsky and Hinton, 2011; Meng et al., 2017; Petscharnig et al., 2017) [3][7][5] might even present a suitable (or at least partially suitable) solution for the problem this project aims to solve and therefore should be further investigated in detail.

In order to set the precise direction of this project, the next step is to identify and choose an approach that has not yet been investigated before.

4 Research methodology

In this chapter, a short overview is given over the methods that are planned to be used or already have been utilised to conduct the research.

4.1 Choosing a data set

As basis of the experimental work, a data set is used in order to search for features in the data and extract them. For this purpose, a suitable data set was searched and identified. The chosen data set [11] comes from Kaggle, the world's largest community for data science, and consists of IP network traffic flows that were collected from the University of Cauca in Colombia. The data was captured in 2017 over a period of six days. The data set contains almost 3.6 million instances (rows), each of them has 87 attributes or features (columns), e.g. source and destination IP addresses and ports, protocol, flow duration, transmission rate and various packet information. Most of them (but not all) are of numeric type. Therefore, the volume and complexity of the data in this set is sufficient to serve as a sample for big data. [11]

4.2 Literature search

The field of deep learning covers a large number of technologies like deep belief networks or autoencoders. A web and literature search will be conducted in order to identify the approaches that are most suitable for the demands of this project, and to analyse their theoretical strengths and limitations. Moreover, an initial literature review was already carried out in order to examine the current state of research and to identify existing approaches and methods that might possibly be related to this project.

4.3 Iterative implementation

Based on the chosen data set and under consideration of the findings of the literature review, a model will be developed that allows to extract features from the data set that can afterwards be used for storage and retrieval purposes instead of the original data. The implementation will be an experimental and iterative process in which different approaches and possible solutions will be investigated and tried out. This includes various types of neural networks with a different topology as well as using different training parameters. Either Alexnet or Tensorflow will be used for the implementation, which are both state-of-the-art technologies that offer a wide range of functions and toolkits as well as a lot of tutorials.

The main challenge is to extract features that provide a high accuracy, which means the extracted features need to describe the original data as precise as possible. In this regard, it is important to ensure that no relevant information is lost, i.e. the extracted features must still contain all vital information from the original data. At the same time, the size of the features should be as small as possible in order to save bandwidth and storage space. Hence, the question is to what extent the size of the data can be reduced and which technique provides the best results in terms of accuracy and saving of storage space.

4.4 Prototyping

In order to evaluate and compare the possible solutions, a prototype will be implemented. Prototyping is a widely-used method to quickly test the suitability of a created piece of software for its intended use. Furthermore, it allows to easily evaluate and compare various potential solutions. In this particular case, the prototype shall use the data set as input, extract features from it and then reconstruct the original data from the features.

The reconstruction error and therefore the accuracy of the model can be measured by comparing the reconstructed data with the original data. When evaluating the solution, it is important to set the achieved accuracy in relation to the size of the features, because the accuracy alone is not meaningful enough. For example, if the accuracy is excellent but the extracted features are almost as big as the original data, then the solution isn't sufficient since it does not fulfil the aim, which is to save storage and bandwidth by using the features. The model that provides the best accuracy in relation to the size of the features will be the winner.

4.5 Network simulation

Once a sufficient solution has been developed, a computer network will be simulated in order to test and evaluate the solution for its actual use, which is the information storage and retrieval. A suitable network simulation tool still needs to be identified which is planned to be done at a later stage of this project.

However, simulating the network brings a lot of advantages over testing in a real network, e.g. because the topology can be arbitrarily chosen and the actual traffic can be easily captured. Furthermore, factors that can negatively affect the measurement data like utilisation of the network must not be considered. Moreover, every conceivable scenarios can be easily simulated, e.g. an empty network or a fully utilised one, in order to vary the network conditions. To realise such scenarios in a real network would be much more effort.

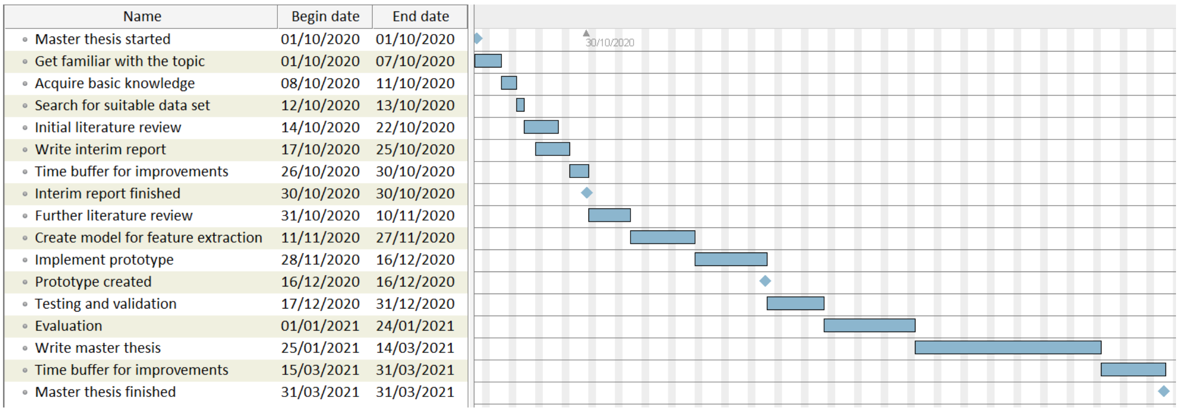
5 Deliverables and outcomes

The expected outcome of the master thesis is to identify a deep learning technique that is capable of extracting suitable features from a data set that can be used for storage and retrieval purposes. Furthermore, it is expected that a model for feature extraction will be developed. The extracted features need to be of high quality, which means they must contain all relevant information of the original data while having a significantly smaller size at the same time.

As deliverable, a functional prototype shall emerge from this project which uses the created model in order to transform an input data set into a set of features that can be transferred and stored. This prototype shall serve as a proof of concept and provide evidence that the developed solution actually works.

6 Project plan

A good and careful planning is decisive for the success of the project. In order to ensure a well-organized project realisation, the time plan shown in figure 6.1 was developed. For that purpose, the project was broken down into smaller tasks that allow an approximate time estimation. The plan covers the whole project cycle from the beginning to the submission of the master thesis. However, the plan doesn't go too much into detail since the precise steps for the research are not yet entirely clear.



Name	Begin date	End date
Master thesis started	01/10/2020	01/10/2020
Get familiar with the topic	01/10/2020	07/10/2020
Acquire basic knowledge	08/10/2020	11/10/2020
Search for suitable data set	12/10/2020	13/10/2020
Initial literature review	14/10/2020	22/10/2020
Write interim report	17/10/2020	25/10/2020
Time buffer for improvements	26/10/2020	30/10/2020
Interim report finished	30/10/2020	30/10/2020
Further literature review	31/10/2020	10/11/2020
Create model for feature extraction	11/11/2020	27/11/2020
Implement prototype	28/11/2020	16/12/2020
Prototype created	16/12/2020	16/12/2020
Testing and validation	17/12/2020	31/12/2020
Evaluation	01/01/2021	24/01/2021
Write master thesis	25/01/2021	14/03/2021
Time buffer for improvements	15/03/2021	31/03/2021
Master thesis finished	31/03/2021	31/03/2021

Figure 6.1: Initial project plan and Gantt chart

As the plan shows, the aim is to be finished with the research tasks and the practical work until the end of January in order to have enough time left to focus on writing a high-quality master thesis.

References

[1] Joe Salomon. Extracting Value from Data with Deep Learning, September 2018. https://missinglink.ai/blog/deep-learning/extracting-value-from-data-with-deep-learning/.

[2] G. E. Hinton and R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

[3] Alex Krizhevsky and Geoffrey E. Hinton. Using Very Deep Autoencoders for Content-Based Image Retrieval. Department of Computer Science, University of Toronto, 2011.

[4] D. Manolescu. Feature Extraction - A Pattern for Information Retrieval. 1998.

[5] Stefan Petscharnig, Mathias Lux, and Savvas Chatzichristofis. Dimensionality Reduction for Image Features Using Deep Learning and Autoencoders. CBMI '17, New York, NY, USA, 2017. Association for Computing Machinery.

[6] Yasi Wang, Hongxun Yao, and Sicheng Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, April 2016.

[7] Q. Meng, D. Catchpoole, D. Skillicom, and P. J. Kennedy. Relational autoencoder for feature extraction. *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 364–371, May 2017.

[8] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50:969–978, July 2009.

[9] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of Big Data 2*, February 2015.

[10] Laurens van der Maaten, Eric Postma, and Jaap van den Herik. Dimensionality Reduction: A Comparative Review. October 2009.

[11] Juan Sebastián Rojas. IP Network Traffic Flows Labeled with 75 Apps | Kaggle. https://www.kaggle.com/jsrojas/ip-network-traffic-flows-labeled-with-87-apps.