# Edge-cloud IoT Data Analytics: Intelligence at the Edge with Deep Learning

Ananda Mohon M. Ghosh, *The University of Western Ontario*

Supervisor: Katarina Grolinger, *The University of Western Ontario*

# Abstract

Rapid growth in numbers of connected devices, including sensors, mobile, wearable, and other Internet of Things (IoT) devices, is creating an explosion of data that are moving across the network. To carry out machine learning (ML), IoT data are typically transferred to the cloud or another centralized system for storage and processing; however, this causes latencies and increases network traffic. Edge computing has the potential to remedy those issues by moving computation closer to the network edge and data sources. On the other hand, edge computing is limited in terms of computational power and thus is not well suited for ML tasks. Consequently, this thesis aims to combine edge and cloud computing for IoT data analytics by taking advantage of edge nodes to reduce data transfer. Feature learning is performed on the edge with deep learning: the encoder part of the trained autoencoder is placed on the edge to extract relevant features. These features are sent to the cloud where the final ML task is performed directly, or the original features are first restored using the decoder part of the autoencoder. In the single-node model, all data are considered together and processed on a single edge node. In the multi-node model, sensors are grouped according to the IoT device locations or according to similarities between sensors, and different groups are processed on different edge nodes. The evaluation was performed on the task of human activity recognition from sensor data. Results show that data can be reduced on the edge up to 80% without significant loss in accuracy.

**Keywords:** IoT, Edge Computing, Deep Learning, Autoencoders, Data Reduction, Human Activity Recognition.

## Summary for Lay Audience

Our daily lives are changing with rapid growth in numbers of smart and internet connected devices, including sensors, mobile, wearable, and other internet of things (IoT) devices. These devices are collecting data and transferring them to the cloud, possibly thousands of miles away, for storing and processing. Then, the results are sent back to the user for process control or decision making. However, transferring IoT data to the cloud is consuming network bandwidth, introducing latencies, and reducing the quality of service. Edge computing has the potential to remedy this situation by processing data close to where they are generated or collected. However, edge computing is not well suited for machine learning tasks as it has limited computation capabilities. Consequently, this thesis investigated merging edge and cloud computing for IoT analytics with the objective of reducing network traffic and latencies. The edge nodes apply machine learning to extract important features from the IoT data and send reduced data to the cloud for the final processing. The evaluation shows that for human activity recognition tasks, data can be reduced up to 80% without significant loss in accuracy.

# Acknowledgements

First of all, I would like to express my profound gratitude to my supervisor Dr. Katarina Grolinger because of her faith in me and allowing me to pursue a research-based degree under her supervision. Throughout this bite of research life, I have been delighted to work with her, especially enjoyed her dedication, patience, and enthusiasm for research. Most importantly, I have always admired her positivity, sense of humor, and style of cheering up when I felt down. It was an absolute honour and privilege to work with such a wise person. I really feel so lucky to be supervised by her, and I am deeply indebted to her, which is not repayable forever. I would also like to show my heartiest gratitude to all the professors who taught me and helped me build the background for this dissertation.

I am really grateful to be a member of my supportive family, especially my dad Gobinda L. Ghosh and my mom Suchittra R. Ghosh who always supported my passion, patronized the carrier I have chosen, and continued to be a source of my inspiration throughout this arduous journey.

I would like to extend my sincere thanks to one of my best friends Shuvra Prakash Sarkar who always motivated me to pursue a Masters's degree.

Last but not least, thanks to all of my friends, especially Deba, Rubel, Arnab, Mithun, Navid, David, Ljubisha, Yifang, Santiago, Tiffany, Chaity, Asif, Grytan, and Sanjoy for being a supportive hand when I was in need. I will always appreciate your generosity.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations, Symbols, and Nomenclature

| | |
|---|---|
| AE | Autoencoders |
| ANN | Artificial Neural Network |
| CNN | Convolution Neural Network |
| CPU | Central Processing Unit |
| CUDA | Compute Unified Device Architecture |
| DBN | Deep Belief Network |
| DL | Deep Learning |
| DNN | Deep Neural Network |
| EC | Edge Computing |
| ECG | Electrocardiography |
| EWS | Early Warning Score |
| FFNN | Feed-forward Neural Network |
| F2C | Fog to Cloud |
| GPU | Graphics Processing Unit |
| HAR | Human Activity Recognition |
| HMM | Hidden Markov Model |
| IOT | Internet of Things |

| | |
|---|---|
| L-CNN | Lightweight Convolutional Neural Network |
| MAE | Mean Absolute Error |
| MAPE | Mean Absolute Percentage Error |
| MEC | Mobile Edge Computing |
| MB | Megabytes |
| ML | Machine Learning |
| MLP | Multi Layer Perceptron |
| MSE | Mean Squared Error |
| M2M | Machine To Machine |
| NN | Neural Network |
| PCA | Principal Components Analysis |
| QoE | Quality of Equipment |
| QoS | Quality of Service |
| RNN | Recurrent Neural Network |
| SDN | Software Defined Network |
| SSL | Smart Street Lamp |

# Chapter 1

# Introduction

## 1.1 Motivation

In recent years, the explosion of sensors, wearables, mobile, and other Internet of Things (IoT) devices has been changing how we live and work. CISCO estimates that the number of connected devices will exceed 28 billion by the year 2022, up from 18 billion in 2017 [2]. More than half of those devices, over 14.6 billion, will be machine-to-machine (M2M) connections. The number of connected devices, together with a flood of data they generate, will increase network traffic. According to Cisco, by the year 2022, global internet traffic will reach 4.8 zettabytes per year, up from 1.5 zettabytes per year in 2017 [2]. Although this will come with positive impacts, including new applications/services and increased use of the existing ones, it will also increase demand on network bandwidth and put pressure on already strained communication infrastructure.

The IoT provides a platform for devices to connect to the Internet and to each other, and enables devices to collect data about their environment. IoT supports smart systems such as smart cities, smart healthcare, smart transportation, and smart energy; however, the realization of these smart systems relies on the ability to analyze these data [3]. On the other hand, most IoT edge devices, such as sensors, do not have computation abilities to perform complex data analytics computations and therefore have been primarily responsible for monitoring the environment and transmitting data to a more powerful system, often the cloud or a centralized system, for storage and processing [4].

Consequently, a typical IoT data analytics involves sending data to the cloud, analyzing it on the cloud, and subsequently delivering results to another device. For example, process monitoring data from a smart factory may be transferred to a data center thousands of miles away where they are stored and processed; then the results are sent back to the same factory for process optimization. This workflow increases not only network traffic, but also data transfer latencies. However, data analytics computation cannot be performed solely on the connected devices as they have limited computation resources. IoT device storage capacity, memory, CPU, and battery life restricts the type of analytics that can be performed on the edge.

*Edge computing* (EC) [5, 6, 7] has been proposed as a way to reduce network traffic and data transfer latencies by physically pushing the computation away from the centralized system and to the edges of the network and the sources of data. Combining edge with cloud computing has the potential to reduce IoT network traffic and associated latencies while still supporting complex data analytics tasks. Performing a part of the computation on the device itself or on a node close to the source of data can reduce data transfer to the cloud and consequently can reduce latencies and improve response time.

By reducing network traffic latencies, edge computing can improve application response time and enhance Quality of Service (QoS). Content delivery is an example of an edge computing application where QoS is enhanced by locating the content close to where it is expected to be delivered. Applications with ultra-low latency requirements can benefit from edge computing as the overall response time can be reduced by shortening network transfer distances.

Although edge computing has been recognized as a powerful approach for tasks such as mobile task offloading [8] and content delivery [9], its use for data analytic has remained limited [7, 10]. Edge Servers [5], servers that reside on the edge of a network, often act as a connection between a private network and the Internet. They can act as intermediaries between the cloud and IoT device by performing computation on data and sending reduced data to the cloud for further processing: for example, Tang *et al.* [11] suggested that edge computing could be used to extract features and reduce the number of features sent to the cloud.

In recent years, deep learning (DL) has demonstrated success in a variety of domains, including image classification [12] and human activity recognition [13]. In the IoT context, DL ability to carry out representation learning and to transform data into hierarchical abstract representations is beneficial for IoT data analytics because it can enable learning good features. A type of DL, a deep autoencoder (AE), is a neural network trained to learn data encoding in an unsupervised manner. Together with encoding, a reconstruction is learned enabling AE to restore its inputs from the reduced encodings, possibly with some loss of information.

This thesis investigates combining edge with cloud computing for IoT data analytics. Edge nodes act as intermediaries between IoT devices and the cloud reducing the quantity of data sent to the could. The encoder part of the trained autoencoder is employed on the edge to create data encodings that are sent to the cloud. The Machine Learning (ML) task on the cloud is carried out in two ways: directly with encoded data, or the original data are first restored with the decoder part of the autoencoder and then used for the ML task.

As IoT data can originate from different sensors and locations, this study explores feature learning from all data fused together, from data grouped by their source locations, and from data grouped according to sensor similarities. Moreover, this thesis investigates the degree of data reduction that can be achieved on the edge without significant loss of ML task accuracy. The evaluation scenarios involve Human Activity Recognition (HAR) from sensor data including accelerometers and gyroscopes. The first scenario uses data from smartphones, while the second scenario includes sensors mounted on different parts of the human body. Results show that the presented approach can reduce data transferred to the cloud up to 80% without a considerable impact on HAR accuracy.

## 1.2 Contribution

The primary objective of this work is to explore combining edge and cloud computing for machine learning, specifically, to investigate the ability of the edge to reduce the quantity of data sent to the cloud. The main contributions of this thesis can be summarized as:

- Edge-cloud architecture for IoT data analytics. The role of the edge is to carry out the preprocessing and to extract relevant features. Autoencoders are used for this task because once an autoencoder is trained, the encoder part for the network can be deployed on the edge to extract representative features. Then, on the cloud, the decoder part of the network can be used to restore the original data, or the ML task can be carried out directly on the reduced features forward from the edge.

- Single and multi-node edge computational models are considered. In the single-node model, data from all sensors arrive to the same node where they are reduced. In contrast, in multi-node models, several edge nodes are involved, and each node is responsible for data arriving from a specific group of sensors. This way, subsets of data are processed in parallel on different edge nodes.

- Location and similarity-based approaches to edge-cloud analytics. In the location-based approach, sensors transmit data to the closest edge node, whereas in the similarity-based approach, sensors are assigned to edge nodes according to sensors similarity. The location-based approach could lead to lower data transfer than the similarity-based approach, but a possible advantage of the similarity-based approach is in carrying out representation learning on similar data.

- Investigation of a degree of data reduction that can be performed on the edge without a significant impact on the accuracy of the machine learning model. Data reduction rates up to 95% have been considered, and their effects on the HAR classification accuracy has been examined.

- Examination of the impact of the sliding window technique on the degree of data reduction and ML tack accuracy. With the sliding window technique, data are buffered on the edge for a specific number of time steps, processed to extract features, and then sent to the cloud for further processing. Different sizes of the sliding windows have been applied.

- Evaluation of the location-based and similarity-based multi-node approaches to data reduction. The results were compared to the single edge node approach as well as to the baseline model involving traditional cloud ML without any data reduction. The experiments were performed on the HAR task with the Mobile Health ( MHEALTH) dataset [14]. The MHEALTH dataset contains recordings of body motion for ten individuals while performing different activities. Results showed that HAR data can be reduced up to 80% without a significant impact on the ML task accuracy.

All experiments were carried out on the human activity recognition task. Two different data sets were used: one from the smartphones and one from sensors mounted on different body parts. On those data sets, the presented approach was able to achieve a high degree of data reduction. The same approach could be used for other IoT tasks. The degree of data reduction, as well as the accuracy of the performed ML task, will depend on the task itself and on the dataset. Further experiments are needed to evaluate how the presented approach would behave with different tasks and data sets.

## 1.3 Thesis Outline

The remainder of this thesis is organized as follows:

Chapter 2 describes the background, which includes Internet of Things in Section 2.1, Edge Computing in Section 2.2, Feed Forward Neural Networks in Section 2.3, Deep Learning in Section 2.4, Principal Component Analysis in Section 2.5 and finally, Evaluation Metrics in Section 2.6.

Chapter 3 discusses the related work. First, Section 3.1 discusses edge computing and its applications related to IoT including smart home and smart city (Subsection 3.1.1), health (Subsection 3.1.1), and manufacturing (Subsection 3.1.3). Next, Section 3.2 introduces the research involving human activity monitoring focusing on the works including edge computing.

Chapter 4 presents the edge-cloud ML system models. First, edge-cloud computing architecture is described in Section 4.1 and data reduction with autoencoders in Section 4.2. Next, the single-node computational model is presented in Section 4.3 and the multi-node computational model in Section 4.4.

Chapter 5 describes evaluation methodology: Section 5.1 presents the evaluation methodology for the single-node and Section 5.2 for the multi-node computational model.

Chapter 6 presents the results and analyzes the findings. Section 6.1 examines the single-node model, Section 6.2 the multi-node model, and 6.4 discusses the overall findings .

Finally, Chapter 7 concludes the thesis and discusses future work.

# Chapter 2

# Background

This chapter first introduces concepts of Internet of Things and edge computing. Next, feed forward neural networks and deep learning are discussed. In the deep learning section, the focus is on autoencoders (AE) because, in this thesis, they are used for data reduction on the edge. Finally, Principal Component Analysis (PCA) is introduced, and evaluation metrics used in this thesis are described.

## 2.1 Internet of Things

Internet of Things (IoT), sometimes referred to as Internet of everything, extends the Internet into the real world and embraces everyday objects as participants in data exchange and communications [15]. Physical objects connected to Internet are able to collect and exchange information with other connected objects without the need for human interaction. Moreover, those devices can be controlled remotely taking advantage of their Internet connection. IoT devices are very diverse; a few examples include smart electricity meters, smart TV, smart appliances, and security systems. The number of IoT devices is increasing rapidly resulting in the explosion of data that needs to be transferred.

Figure 2.1 illustrates the IoT landscape. Although the figure only includes some of the end devices (vehicles, buildings, industrial systems, etc.) and a few applications (smart city,

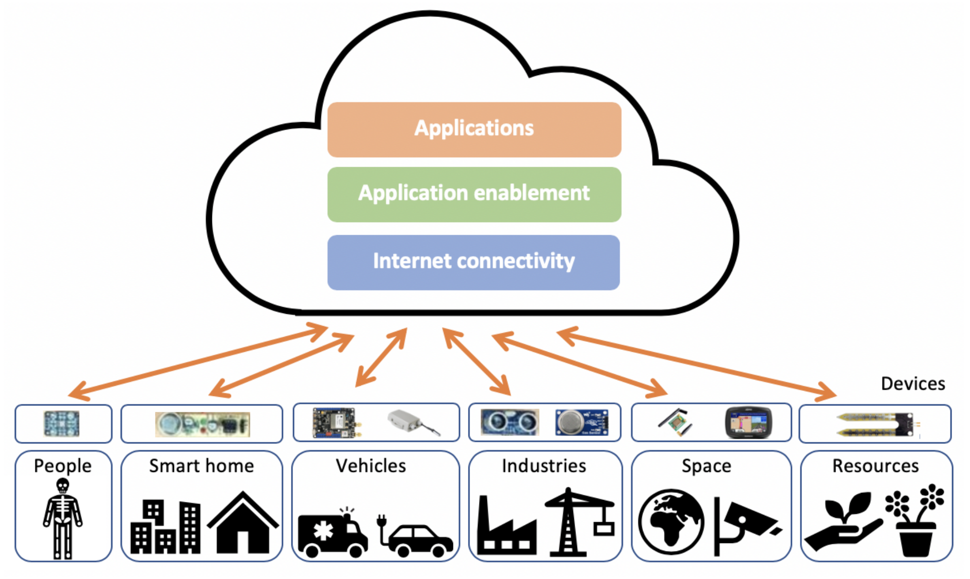

Figure 2.1: Internet of things.

agriculture, etc.), the diversity of the possibly involved entities is well demonstrated. The common characteristics across the landscape include the Internet connection, some type of sensors interacting with the environment, and the software application.

In the IoT vision, the autonomous and secure connection is established among real-world devices and software applications enabling the exchange of data and remote device control [16]. Consequently, IoT connects physical devices with the virtual world [16]. Occasionally, logic, software, and task processing facilities can be integrated into physical devices, and those devices can operate somewhat as embedded systems [17]; however, as resources on the physical devices are limited, only very simple operations can be performed on end devices. With the emergence of new sensors and new IoT applications, the role of IoT in our society has been growing resulting in increased network traffic. As the transfer of large data over the network is causing latencies, there is a need to investigate ways of reducing the network traffic while maintaining IoT system capabilities.

## 2.2 Edge Computing

Centralized infrastructure systems, such as the cloud or the enterprise server, store data, execute business logic, and perform data analytics tasks far away from the end users and data sources. They offer great advantages of immense computation capability, high scalability and reliability, pay-as-you-go billing model, and low initial cost. However, with zettabyte-sized traffic and the explosion of connected devices, transferring all data to the cloud for processing is not practical or even feasible. Edge Computing (EC) has emerged as a way of dealing with these challenges by pushing computation to the edges of the network and sources of data [7].

Fog computing shares many characteristics with edge computing. Occasionally, the terms “fog” and “edge” are used interchangeably [18]; but edge computing focuses more on nodes closer to IoT devices, whereas fog can include any resource located anywhere between the end device and the cloud. In this work, we use the term “edge” to refer to the end devices themselves, such as sensor nodes and smartphones, as well as computation nodes located close to the network edge such as the edge servers.

The key idea behind the edge computing is to reduce the network traffic by bringing computation closer to the data sources. Figure 2.2 show the edge computing role in an IoT system: the edge acts as an intermediary between end devices and the cloud with the objective of reducing latencies and network traffic. Examples of edge computing use cases include autonomous vehicles, healthcare devices, retail advertising, smart speakers, and video conferencing [19].

The edge computing has been investigated extensively in mobile computing: mobile edge computing offloads computation and data storage to the edge (e.g., base stations) to reduce network latencies, improve user experience, reduce battery consumption, and introduce location-awareness [20]. EC is especially suitable for applications with ultra-low latency requirements and for content delivery and caching [21].

Even though edge computing provides advantages of reduced network traffic, computing resources available on the edge are not comparable to those present in the cloud. Thus, computationally intensive tasks, such as ML, are not well suited for edge devices. Nevertheless,

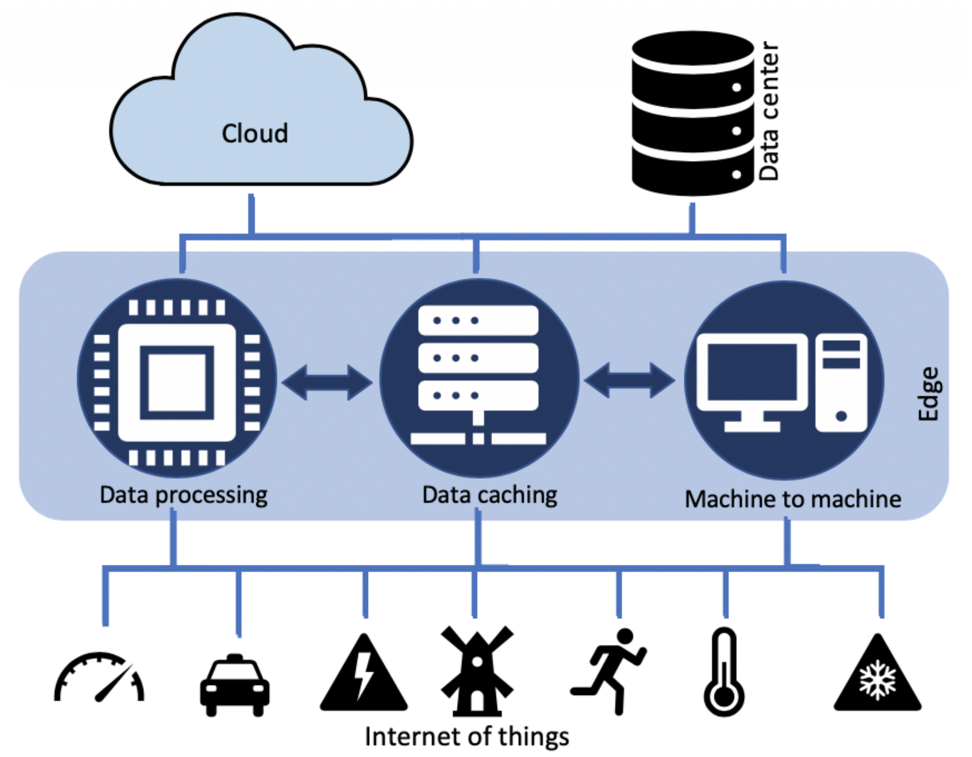

Figure 2.2: Edge Computing.

the edge can supplement cloud computing and perform part of the computation, consequently reducing network traffic and latencies. In this thesis, edge computing is combined with cloud computing for IoT data analytics.

## 2.3 Feed Forward Neural Networks

Neural networks (NN) [22] are machine learning models inspired by neurons in the human brain. Like a human brain, NNs are composed of neurons, also referred to as perceptrons, connected with synapses (weights) which carry signal between neurons. Feed Forward Neural Network (FFNN) is one of the oldest and simplest NN architectures [23]. FFNN consists of an input layer, an output layer, and one or more hidden layers: Figure 2.3 shows FFNN with three hidden layers. Neurons in each layer are fully connected to all neurons in the previous layers, and there are no connections between neurons in the same layer. The number of neurons in

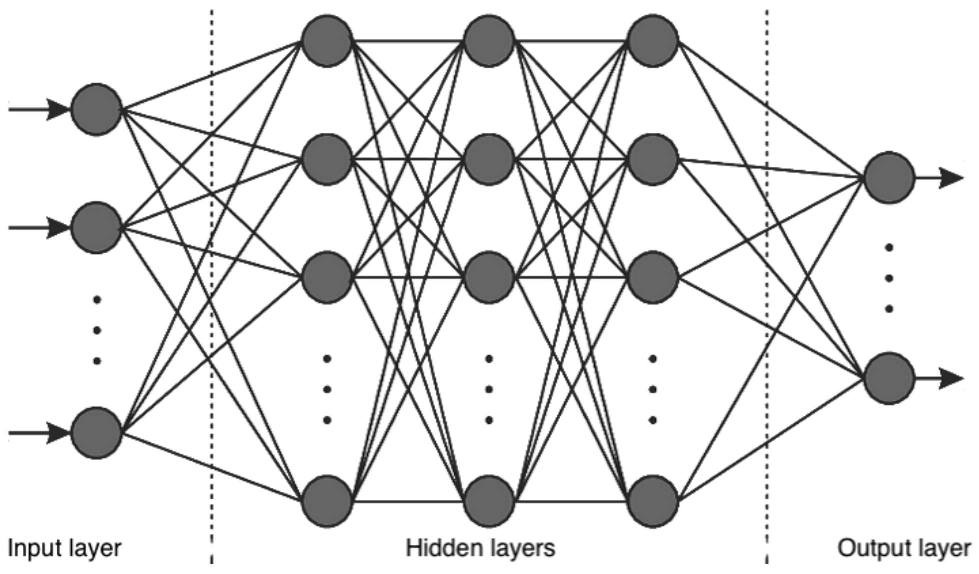

Figure 2.3: Feed Forward Neural Network.

the input layer is equal to the number of input features, while the number of output neurons corresponds to the number of classes or the number of function outputs.

The FFNN training process starts by initializing weights to random values. Next, in the feedforward pass: an input sample represented as a vector with the number of elements corresponding to the number of input features is passed to the input layer. Information flows from the input layer, through a hidden layer(s), to the output layer, and activation for each neuron in the network is calculated. Once the output neuron activation is calculated, the error is determined by comparing the network output with the desired/known output from the training sample. This error is backpropagated [24] through the network, and the weights are updated according to the optimization method to reduce the error for that specific training sample. This forward pass and the backward pass are repeated for all samples in the training set: this is referred to as an *epoch*. Typically, NN requires several epochs for the weights to converge. After the weights converge, the NN is ready for use.

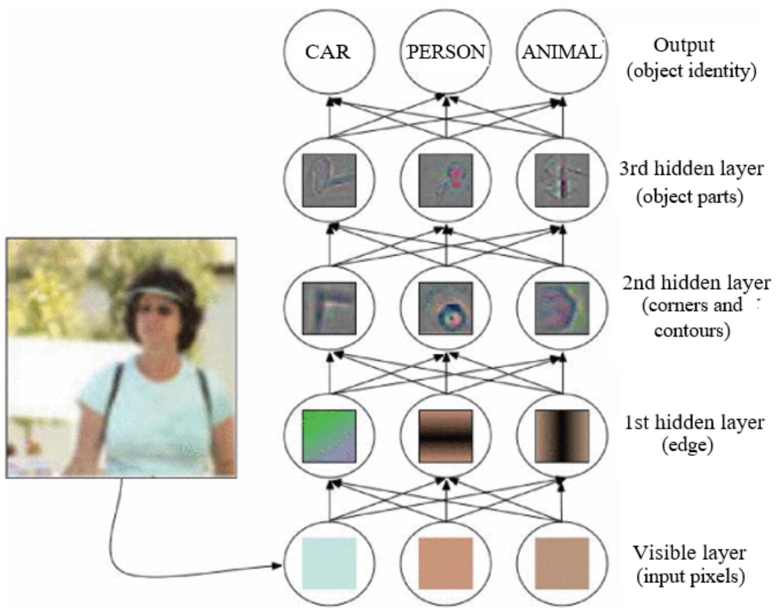

Figure 2.4: Deep Learning [1].

## 2.4 Deep Learning

Deep learning (DL) is a class of machine learning algorithms that use a cascade of multiple layers, with each one performing a non-linear transformation [25]. In recent years, DL has gained popularity because it demonstrated an ability to learn complex models and perform representation learning [3]. Figure 2.4 illustrates the deep learning process on the object recognition task. Each layer learns a specific feature: edges, corners and contours, and object parts. The final layer performs the ML task using the learned features. Deep learning uses data representations rather than explicit data to perform learning: data are transformed into hierarchical abstract representations that enable learning based on the features.

DL architectures are versatile, and a few popular examples include autoencoders, convolutional, and recurrent neural networks. Because of the ability to extract global relationships from data and reliance on high level abstractions, deep learning can be used for both supervised and unsupervised learning. This thesis uses unsupervised learning, specifically autoencoders, to reduce data sent from the edge to the cloud.

Autoencoders (AE) [26] are a subcategory of deep learning approaches used for learning data representations (encodings) in an unsupervised way. Essentially, an autoencoder is a neural network (NN) that learns to reconstruct its inputs; bottleneck NN layers prevent it from merely copying the input to the output and force it to learn data representations.

As illustrated in Figure 2.5, an AE consists of Encoder and Decoder, each one possibly composed of several stacked layers. The encoder part of the network is responsible for reducing dimensionality (encoding); therefore, the number of neurons typically reduces starting from the input layer to the last encoder layer. In contract, the decoder part is responsible for reconstructing the input signal from the encoded values and thus typically consists of layers with a gradually increasing number of neurons.

For example, if a input vector $x_1, x_2, ..., x_n$ is passed through the AE, the input vector is compressed through the hidden layers which have a fewer number of neurons than the input layer, into lower dimensional space and then uncompressed to reproduce the $x'_1, x'_2, ..., x'_n$ as the output.

An AE can be used for noise removal and anomaly detection, but they often serve as a preprocessing step for another ML task [3]. AEs have great potential in the IoT context because they can carry out representation learning by transforming data into hierarchical abstract representations that enable learning good features [27]. Once an AE is trained, the encoder network can be used for dimensionality reduction by taking encoder outputs (encodings) as inputs to another ML model.

In this thesis, the encoder part of the trained AE is deployed on the edge to reduce dimensionality before the data are sent to the cloud. Moreover, the decoder is employed on the cloud to reconstruct the original signal.

## 2.5 Principal Component Analysis

Principal Component Analysis (PCA) [28] is a widely used linear dimensionality reduction technique. It uses orthogonal transformations to convert a set of possibly correlated features

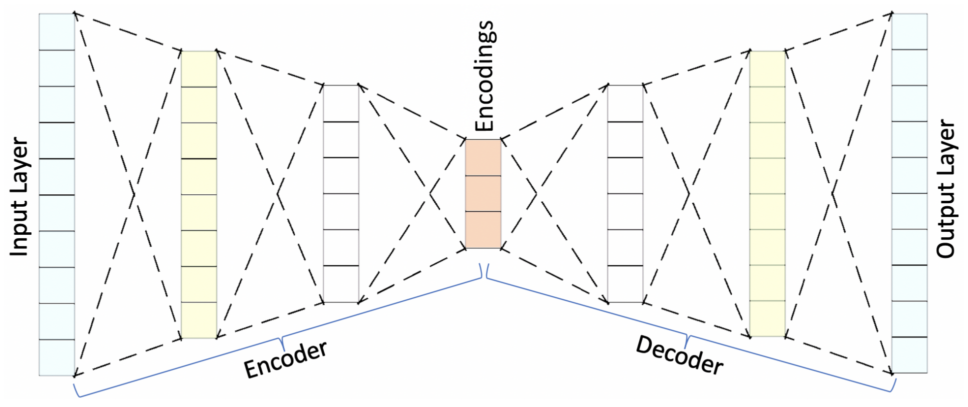

Figure 2.5: Autoencoder.

into a set of linearly uncorrelated features, referred to as principal components. The first principal component explains the largest part of the data variation, and each following component explains the next highest variance under the constraint that it is orthogonal to the preceding components. Dimensionality reduction is achieved by using only the first $n$ principal components.

Let us consider a data set $X$ of dimension $p \times q$ were $p$ is the number of observations and $q$ the number of features. This $X$ matrix is first centered by subtracting the mean $\mu$ from $X$ resulting in $X'$ matrix. Then, the covariance matrix is computed, and the eigenvalues and eigenvectors of this covariance matrix are calculated. Eigenvectors form the columns of $V$ matrix; thus, $V$ dimension is $q \times k$ where $q$ represents features, and $k$ is the eigenvector dimension. Then, PCA projection of $X'$ can be calculated as:

$$Z = X'V \tag{2.1}$$

Here, $Z$ dimension is $p \times k$, where $p$ is the number of samples, and $k$ is the number of components. Dimensionality reduction is achieved by only using $m$ components, where $m < k$ [29].

When dimensionality reduction is carried out with PCA, the original feature values ($q$ feature space) can be restored from $k$ principal components by taking advantage of matrix $V$.

Specifically, $X'$ can be restored as:

$$X' = ZV^T = X'VV^T \tag{2.2}$$

Next, mean $\mu$ needs to be added to reconstruct original matrix $X$:

$$X = X'VV^T + \mu \tag{2.3}$$

If only $n$ principal components are used, the original signal will be restored with some error. In this thesis, data reduction with autoencoders is compared to data reduction using PCA.

## 2.6 Evaluation Metrics

The presented edge-cloud architecture for IoT data analytics was evaluated on the human activity recognition (HAR) task. As HAR is a multi-class classification problem where activities are categorized as standing, walking, and similar, classification metrics have been used [30]. Table 2.1 shows the confusion matrix: Actual Class stands for the known labels from the dataset, and Predicted Class stands for the output of the classification model. Remaining values in the table are as follows:

- True positives (TP) - the number of samples correctly identified as belonging to the positive class
- True negatives (TN) - the number of samples correctly identified as belonging to the negative class
- False positives (FP) - the number of samples incorrectly identified as belonging to the positive class

- False negatives (FN) - the number of samples incorrectly identified as belonging to the negative class

Table 2.1: Confusion Matrix.

| | | **Actual Class** | |
|---|---|---|---|
| | | Positive | Negative |
| **Predicted Class** | Positive | TP | FP |
| | Negative | FN | TN |

With the help of the values from confusion matrix, the classification Accuracy, Precision, and Recall can be calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.4}$$

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

Accuracy is the proportion of the samples correctly classified, precision is the proportion of positive identifications that was correct, and recall is the proportion of actual positives correctly identified as positives.

In addition to classification accuracy, the performed experiments also evaluate how similar is the reconstructed sample to its corresponding original sample. This is done with *Mean Squared Error* (MSE) and *Mean Absolute Error* (MAE):

$$MSE = 1/N \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

$$MAE = 1/N \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

where $y_i$ and $\hat{y}_i$ are the actual and the corresponding predicted values, and $N$ is the number of observations.

When autoencoders reduce data and then reconstruct the original sample, there will be an error depending on how much data was reduced in the autoencoder bottleneck. This error is evaluated with MSE and MAE.

# Chapter 3

# Related Work

This section first discusses related edge computing work including edge computing applications in smart homes and cities, healthcare, and manufacturing. The edge computing approach presented in this thesis was evaluated on the human activity recognition task; therefore, this chapter also reviews related work in human activity monitoring.

## 3.1 Edge Computing

Edge computing has been gaining popularity, especially with applications that require fast response time and those with limited bandwidth because it locates computation close to the data sources. Applications of edge computing including smart street lamps [31], face identification [32], smart manufacturing [33], and vehicular networks [34] have demonstrated great success and prompted further investigations.

Wang *et al.* [35] presented a survey on mobile edge networks focusing on computing-related issues, edge offloading, and communication techniques for edge-based systems. The use cases highlighted in their study include IoT, connected vehicles, content delivery, and big data analysis. At the same time, Wang *et al.* [35] identified real-time analytics as one of the open challenges. Similarly, Abbas *et al.* [21] surveyed mobile edge computing and also identified big data analytics as a future research direction. While Wang *et al.* [35] and Abbas *et al.* [21] examined mobile edge computing, El-Sayed *et al.* [7] focused on IoT applications of edge computing. They compared the characteristics of cloud, multi-cloud, fog, and edge computing and identified low bandwidth utilization and latencies as the main edge computing advantages. Mao *et al.* [36] see edge computing as a key enabling technology for realizing the IoT vision and, similar to Wang *et al.* [35] and Abbas *et al.* [21], recognize data analytics as one of the future research directions in edge computing.

Discussed surveys [7, 21, 35, 36] note the potential of edge computing in data analytics and point out the importance of edge computing in IoT for handling the rapid increase of the number of connected devices. This thesis contributes to employing edge computing for data analytics by combining edge and cloud computing for the delivery of ML applications.

Several studies present various scenarios with accompanying edge-based architectures demonstrating edge computing capabilities. Sinaeepourfard *et al.* [37] proposed the fog to cloud (F2C) data management architecture incorporating the data preservation block to provide faster data access than the cloud. To illustrate the possible benefits of the proposed architecture, they calculated the potential reduction in the data transfer volume and latency decrease, taking the city of Barcelona as an example. Sinaeepourfard *et al.* did not run real-world experiments. Jararweh *et al.* [38] proposed a hierarchical model composed of mobile edge computing servers and cloudlets, small clouds located close to the edge of the network. Their experiments consisted of simulation scenarios; varying numbers of requests were generated with the objective of demonstrating how the offloading impacts the power consumption and the incurred delay. While Sinaeepourfard *et al.* [37] and Jararweh *et al.* [38] demonstrate potentials of edge computing, this thesis is concerned with embedding intelligence in the edge for data analytics tasks.

Edge computing has been used to reduce network traffic in a variety of applications, while video compression has been a common way of dealing with video downloads, uploads, and streaming. Video compression is based on the understanding that information between consecutive frames changes very slowly. For example, the background may not need to be encoded for each frame but can be reused. As the name indicates, video compression targets specifically videos and takes advantage of relationships between video elements. In contrast, our approach is designed for IoT data with the objective of reducing network traffic specifically for machine learning applications. In our approach, autoencoders find relationships between readings within the same timestep and well as between consequent timesteps through the sliding window approach.

In addition to studies investigating edge computing architectures, potentials, and advantages in general scenarios, a number of studies investigated applications of edge computing in specific domains. The three most commonly discussed areas of edge computing applications are smart homes and cities, healthcare, and manufacturing; therefore, the following subsections discuss edge computing applications in those domains.

### 3.1.1 Edge Computing in Smart Home and Smart City

Smart home and city scenarios are one of the most commonly discussed use cases and applications of edge computing [7]. While the smart grid, smart traffic lights, and smart vehicles are sometimes considered separately [7], here we consider them as use cases within the smart city domain.

Mohammad *et al.* [39] examined possibilities of service-oriented middleware for cloud and fog enabled smart city services. They did not discuss specific smart city services but focused on the middleware. Their experiments demonstrated the benefits of edge computing in terms of response time.

Tang *et al.* [11] presented a hierarchical fog computing architecture for the support of connected devices in smart cities. In addition to the hierarchy of fog nodes, the proposed model includes the cloud as the top layer. The evaluation was performed on an event detection task in a smart pipeline monitoring system: the preliminary results demonstrated the feasibility of the proposed architecture. Similar to Mohammad *et al.* [39] and Tang *et al.* [11], this dissertation also employs both edge/fog and cloud, but differs from theirs in that it also includes an extensive evaluation of the presented edge-cloud architecture.

A fog-enabled real-time traffic management system investigated by Wang *et al.* [40] offloads computation to the fog nodes to minimize average response time for events reported by vehicles. Their system consists of three layers: the cloud, cloudlet, and fog layer. Vehicles act as fog nodes, cloudlets are assigned to the city regions, and cloud servers act as the integration system if needed. Conducted simulation experiments are based on the real-world traces of taxies and focus on exploring the effects of the number of fog nodes and service requests. Wang *et al.* [40] focused on message passing and processing while this thesis deals with machine learning for IoT.

The study by He *et al.* [41] presented a multi-tier fog computing model for large-scale data analytics services in smart cities. The multi-tier fog consists of ad-hoc fogs and dedicated fogs with opportunistic and dedicated computing resources, respectively. Offloading, resource allocation, and Quality of Service (QoS) aware job admission were designed to support data analytics and maximize analytics service utilities. In their experiment setup, ad-hoc nodes are lower resource devices such as Raspberry PIs and desktops, while higher resource servers are used as dedicated nodes. They evaluated the proposed model on the classification tasks: the results demonstrated that fogs can improve the performance of smart city services. While the work of He *et al.* [41] deals with the fog architecture, our study takes advantage of both edge and cloud for the ML task.

Jia *et al.* [31] proposed a Smart Street Lamp (SSL) system based on fog computing for smart cities to reduce maintenance periods, decrease energy consumption, providing fine-grain control, and reducing theft. The evaluation demonstrated that the SSL system is capable of self-understanding various static, pre-defined states, and consequently able to improve issue detection and maintenance. Their experiments focused on the overall features of the system and did not specifically consider and compare the advantage of fog servers over cloud servers.

Hossain *et. al* [42] presented an edge computing framework for enabling situation awareness in a smart city. In their approach, a part of the processing is carried out on edge servers, and the computation is finalized on the cloud server by aggregating information forwarded from the edge nodes. Finally, the end user is presented by the situation awareness image-like view. Like this thesis, Hossain *et. al* employed edge nodes to carry out part of the computation; however, while Hossain *et. al* addressed a specific scenario (situation awareness), this thesis is concerned with employing edge computing for machine learning tasks.

### 3.1.2 Edge Computing in Healthcare

This thesis evaluates the presented edge-cloud approach on the use case of human activity recognition (HAR). Applications of HAR include assisted living, healthcare monitoring, fitness tracking, and work assessment. As many HAR applications fall into the healthcare category, it is important to review edge-based systems in healthcare.

Rahmani *et al.* [43] presented a fog-assisted architecture for smart e-Health which embeds intelligence between sensors and the could. In a traditional healthcare IoT system, gateways are used to translate between protocols and send data to the cloud. Rahmani *et al.* exploit the position of these gateways to offer computing services such as storage, real-time processing, and data mining. The fog computing concepts are employed by using gateways to form a geo-distributed intermediary between edge nodes and cloud. Rahmani *et al.* demonstrate the proposed approach with a prototype Early Warning Score (EWS) health monitoring system for estimating the degree of illness and the risk of the patient deterioration in a hospital. In their study, fog nodes are quite powerful and thus able to handle data filtering, compression, fusion, and analysis with only minimal data sent to the cloud. This dissertation, on the other hand, still performs a large part of the computation on the cloud.

Ritrovato *et al.* [44] also dealt with healthcare and proposed an edge-cloud computing architecture for real-time anomaly detection from sensor data streams. In their work, the main role of the edge nodes is to collect data from the sensor layer, convert it into different representations, and send it to the cloud. While Ritrovato *et al.* focus on stream processing algorithms and use edge devices solely for representation transformation, this dissertation deals with ML algorithms and combines edge and cloud to carry out ML tasks.

Chen *et. al* [45] proposed edge-cognitive computing-based smart healthcare system. The system uses cognitive and edge computing to monitor and analyze the physical health of users. Machine learning and deep learning are the key technologies of their cognitive engine responsible for processing and analyzing data. While Chen *et. al* position the ML task on the edge nodes, in our work, the edge and cloud are combined for ML.

Wang *et. al* [46] proposed HealthEdge, a task scheduling approach for edge-cloud healthcare systems with health emergency considerations. HealthEdge assigns different priorities to tasks and determines if the task should be executed on the edge or cloud based on human health status data. The main objective of their approach is to reduce processing time for medically urgent tasks.

Overall, in the healthcare domain, the potential of edge-cloud systems has been recognized, and various studies considered edge computing for different healthcare tasks. However, the reviewed studies consider general purpose computing, while this thesis considers combining edge and cloud for machine learning tasks. Chen *et. al* [45] did consider ML, but, in their work ML is located on the edge nodes while in our work the edge and cloud are combined for ML.

### 3.1.3 Edge Computing in Manufacturing

In manufacturing, edge computing offers not only the benefit of reduced network traffic and latencies but also enables the company to keep data on premises, therefore, reducing exposure to cloud-related privacy and security threats.

Li *et al.* [33] proposed a manufacturing inspection system based on deep learning and fog computing for defect detection in large factories with big data. In their system, production images are captured by cameras and sent to the convolutional neural network (CNN) for defect detection. In contrast to traditional CNN, where all layers are located on a single computing node, in the architecture proposed by Li *et al.*, lower-level layers are located on the fog nodes and higher-level layers on the server. Performed experiments demonstrated high defect detection accuracy, decreased load on the central servers, and reduced overall computation time; however, latencies and communication overhead were not analyzed.

Chen *et al.* [47] presented an edge computing architecture for IoT-based manufacturing and analyzed four aspects of edge computing including edge devices, network communication, information fusion, and integration with cloud computing. They highlighted edge computing benefits of agility, real-time processing, and autonomy in intelligent manufacturing. Also, Chen *et. al* [47] recognized that edge devices could be used for artificial intelligence and machine learning, but they did not provide any specifics on how this would be achieved, nor they consider combining the edge and cloud for ML.

A hybrid computing solution for smart manufacturing proposed by Li *et. al* [48] consists of four layers: device layer, edge computing, cloud system, and Software Defined Network (SDN) layer. The focus of their work is on resource scheduling strategy in the edge-cloud system to achieve low latencies. Experiments demonstrated that the proposed strategy reduces latencies in comparison to cloud computing and edge computing without scheduling. Like this thesis, the work of Li *et. al* [48] considered AI tasks and their location in the edge-cloud system. The difference is that Li *et. al* used the scheduling algorithm to find the node for the AI task allocation, while this thesis is concerned with combining edge and cloud nodes for the ML task.

Wu *et. al* [49] presented a fog enabled framework for monitoring machine conditions and for predictive analytics. In their work, the cloud is responsible for building (training) the predictive model, while the local node is responsible for applying the trained model to new data. By locating the trained model on the edge, prediction latency is reduced. The streaming data is sent to the cloud for ML training. The presented case study demonstrated the ability of the proposed architecture on the task of predicting tool wear and investigated the accuracy of the achieved prediction but did not analyze the computation benefits of edge computing. In Wu *et. al* [49] approach, the cloud is not involved in the prediction/inference stage while in our work, the cloud is still involved as data from different edge nodes needs to be integrated for the final ML task. Moreover, our study reduces the quantity of data sent to the cloud.

Reviewed edge-computing studies in manufacturing demonstrate benefits highlighting reduced latencies and reliance on the on-site facilities. Moreover, some consider AI and/or ML [48, 49] in their case studies. Nevertheless, our study is different as it merges cloud and fog nodes to achieve a single task, and it analyzes the reduction in network traffic.

## 3.2 Human Activity Monitoring

As this thesis evaluates the edge-cloud data analytics approach on sensor-based human activity recognition (HAR), it is important to review recent works from this category.

Given the fact that deep learning has been quite successful and extensively used for HAR [50], the work of Wang *et al.* [50] surveyed deep learning approaches for activity recognition; they highlighted the importance of model selection and the significance of preprocessing including the sliding window technique. Zdravevski *et al.* [51] specifically focused on feature engineering for HAR. This thesis incorporates the sliding window technique from Wang *et al.* [50] work and, to addresses the feature engineering challenge discussed by Zdravevski *et al.* [51], we take advantage of autoencoders' representation learning characteristics.

Uddin [52] proposed a wearable sensor-based activity recognition system for smart healthcare. Data from wearable sensors including magnetometer, accelerometer, gyroscope, and electrocardiography (ECG) sensors, are sent to the edge node, i.e., personal computer or laptop, for activity recognition. The recognition is carried out on the edge node using the Recurrent Neural Network (RNN) with Graphics Processing Unit (GPU) acceleration. In their experiments, proposed RNN outperformed Deep Belief Network (DBN) and Hidden Markov Models (HMM). Like Uddin [52], we also use edge nodes and RNNs; however, while Uddin only used the edge nodes (without the cloud), our work combines the edge and cloud. Moreover, this thesis also analyzes network traffic reduction achieved through edge computing.

There were also studies investigating human detection and identification from pictures and videos. Bellavista *et. al* [53] studied dynamically extending edge computing to mobile devices to exploit the crowd for mobile computing. Their main focus is on investigating the interconnections among continuously moving geographically distributed nodes, while video analysis for face recognition is only mentioned as a use case.

Hu *et al.* [32] similarly used fog computing first to detect a face in an image and then to identify the individual. Like our study, the work of Hu *et al.* [32] used edge nodes to extract features, but Hu *et al.* dealt with images and face identifiers, while our work is concerned with IoT sensor data. The presented experiments [32] demonstrated a reduction in network traffic and response time. While their approach allows for a single degree of data reduction on the edge, our work considers different degrees of data reduction and their impact on accuracy.

Nikouei *et al.* [54] also investigated human detection from surveillance video streams: they proposed a lightweight Convolutional Neural Network (L-CNN) to detect pedestrians with edge devices. While our work uses a combination of edge and cloud nodes, Nikouei *et al.* [54] use only edge nodes.

Uddin [52], Hu *et al.* [32], and Nikouei *et al.* [54] used edge computing for different human activity-related tasks. Uddin [52] and Nikouei *et al.* [54] only used edge nodes while Hu *et al.* [32] combined edge and cloud nodes. Moreover, Hu *et al.* employed edge nodes for feature extraction as we do in our work. However, while they are concerned with videos and one degree of data reduction, our work deals with sensor data and compares different degree of data reduction on the edge.

# Chapter 4

# Edge Cloud ML System Models

This chapter first describes the edge-cloud computing architecture for machine learning with IoT data and data reduction with autoencoders for edge-cloud systems. Next, the single-node edge-cloud computation model will be discussed, followed by the multi-node model.

## 4.1 Edge-cloud Computing Architecture

The edge-cloud architecture for data analytics is depicted in Figure 4.1. Similar to any other edge-based system, data from sensor-equipped devices such as smartphones, activity monitors, and smart meters, are transferred to the edge nodes for further processing. Pure edge computing systems perform complete computation on the edge nodes, whereas typical edge-cloud systems carry out task-specific computation on the edge and possibly connect to the cloud for integration purposes [40].

Sensors are capable of high-frequency sampling: for example, voltage sensor can record thousands of readings per second. Because of this, the quantity of data that need to be transferred is very large. Attempting to move all these data to the cloud for processing will result in high latencies and increased network traffic. Moreover, ML with such large data sets is challenging, time consuming, and sometimes infeasible. Therefore, the role of the edge nodes in the edge-cloud architecture for data analytics presented in Figure 4.1 is to reduce the quantity

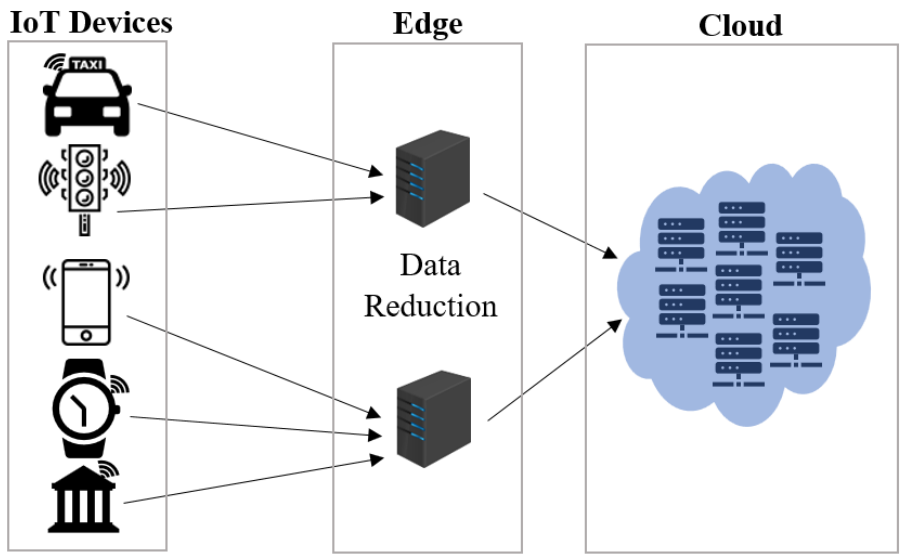

Figure 4.1: Edge-cloud computing architecture for data analytics.

of data transferred to the cloud in a way suitable for machine learning. While other edge computing solutions also reduce data transfer, the solution presented here focuses on data reduction for ML tasks.

The two main categories of data reduction techniques in ML are *dimensionality reduction*, which reduces the number of variables under consideration, and *instance selection*, which selects a data subset for ML [3]. This thesis uses a dimensionality reduction-based technique. After the data are reduced on the edge layer, they are sent to the cloud for ML tasks. Data from different sensors may be sent to different edge nodes, as illustrated in Figure 4.1, but all nodes forward the reduced data to the centralized location. In this way, ML models residing on the cloud can take advantage of the data coming from different places and through different edge nodes. Specific tasks could possibly be carried out on the edge nodes, but those would only have access to data from a subset of sensors.

## 4.2 Data Reduction with Autoencoders

For data reduction on edge nodes, this thesis uses autoencoders. As already mentioned, autoencoders are capable of dimensionality reduction by learning hierarchical data representations. As illustrated in Figure 4.2, autoencoders take input data and process them through several hidden layers. The number of neurons in the hidden layers is smaller than the number of neurons in the input layer, which forces an autoencoder to learn an internal representation of data. An autoencoder consists of two parts: the encoder part compresses data by transforming it into abstract representation (encodings), and the decoder part is responsible for reconstructing the original data from the abstract representation. The inner layers of the autoencoder can be used as features for ML tasks.

To use an autoencoder for data reduction in the edge-cloud architecture, the encoder part of the trained model is located on the edge, and the decoder part is on the cloud. This way, when high-dimensional data ahttps://www.overleaf.com/project/5e31eb096847d40001ff38d5rrive at the edge node, they are reduced to a smaller number of dimensions according to the encoder architecture. After these data are sent to the cloud, they can be directly used for ML tasks, or the original signal can be reconstructed through the decoder part of the autoencoder located

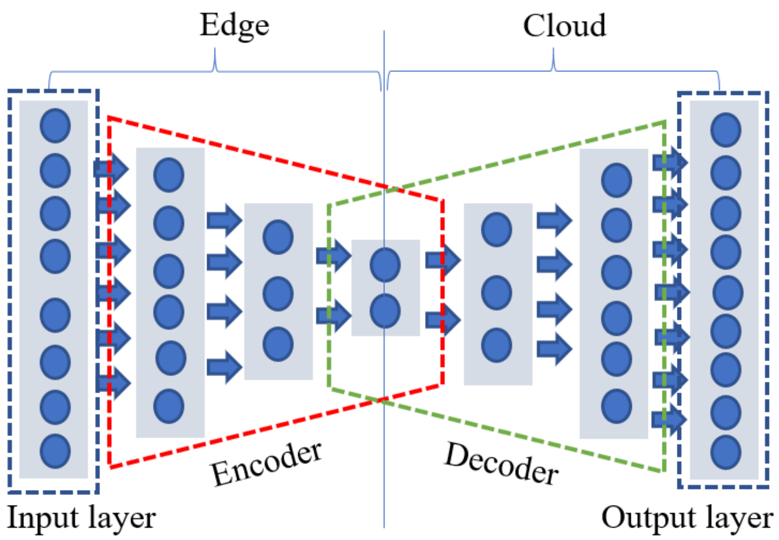

Figure 4.2: Autoencoder for dimensionality reduction in edge-cloud architecture.

on the cloud and then used for ML tasks. Note that this is the processing layout used after the autoencoder is trained. Because autoencoder training is computationally expensive, it still needs to happen on the high-resource nodes such as the cloud or GPU-enabled devices. An example of this comparatively very high resource and extremely high dimensional data reduction is could be video. Which could be achieved by recurrent convolutional networks based autoencoder [55].

Features can also be reduced by using PCA, but PCA carries out linear transformations while AE performs non-linear transformations. Consequently, this thesis employs AEs for data reduction on edge. Experiments compare PCA and AE in terms of data reduction rates and accuracy.

## 4.3 Single-node Edge-cloud Computation Model

The single-node edge-cloud computation model involves only one edge node for all edge-based computation, and this single node is responsible for receiving data from multiple sources and/or sensors. For the single-node edge-cloud model, this thesis considers three computation scenarios, as illustrated in Figure 4.3. Scenarios 1 and 2 employ edge-cloud data analytics, whereas Scenario 3 is a traditional cloud-based analytics scenario included solely for comparison purposes.

**Scenario 1:** Data from sensors are sent to the edge node where data reduction is performed using the encoder part of the trained autoencoder. Reduced data are sent to the cloud where machine learning is carried out directly with the compressed data. In this scenario, the primary concern is the accuracy of the machine learning task; for example, in the case of classification, accuracy could be expressed through measures such as precision and recall. Note that autoencoders are used because they perform non-linear dimensionality reduction; nevertheless, other techniques could be used, such as PCA, as will be demonstrated in the experiments.

**Scenario 2:** As in Scenario 1, data from sensors are sent to edge nodes where reduction is

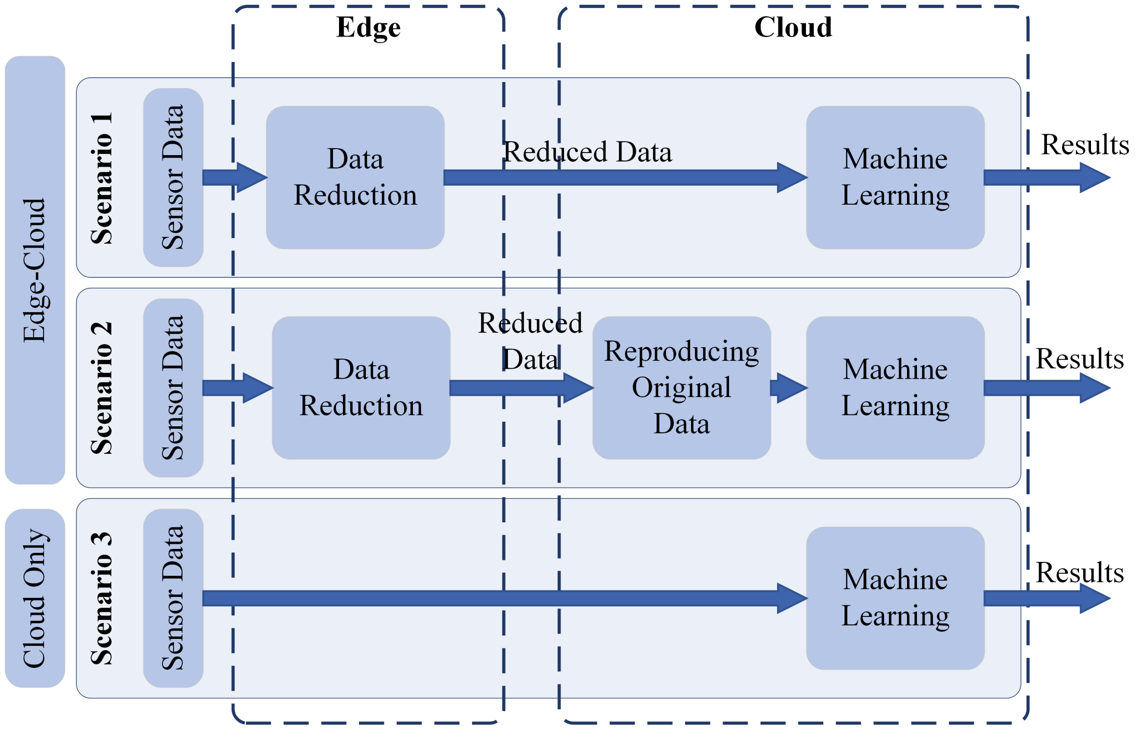

Figure 4.3: Single-node edge-cloud computation model.

performed, and the reduced data are sent to the cloud. Instead of carrying out the ML task directly on the reduced data, as it was done in Scenario 1, original data are reconstructed using the decoder part of the autoencoder, and ML uses the reconstructed data. This scenario transfers the same quantity of data from the edge to the cloud as Scenario 1. ML is performed on a larger quantity of data since original data are reconstructed; however, data reconstructing can provide more flexibility with respect to the features used for different ML tasks. As in Scenario 1, the accuracy of the final ML task is important. Additionally, in this scenario, the accuracy of the reconstructed signal is a major concern. If the autoencoder's inner layers have overly low numbers of neurons; in other words, an attempt was made to compress data too much, the accuracy of the reconstructed signal will be affected. Examples of error measures for reconstruction accuracy include MSE and MAE. If the reconstructed data are of low quality, the accuracy of the final ML task will be reduced. Note that, in this scenario the reproduction the data on cloud needs extra time, which will introduces latencies comparatively with scenario 1 since its performing extra execution in the cloud.

**Scenario 3:** This is a pure cloud computing scenario: data are sent directly from sensors to the cloud, and ML is performed with these data. It is included here only as a baseline for comparison with the other two scenarios.

Of course, with ML it is important to keep the accuracy of the final machine learning task as high as possible. Because the main objective of the presented architecture is to reduce network traffic and latencies, the amount of data that can be reduced on the edge needs to be considered.

When autoencoders are used for data reduction, the number of selected features is determined by the hidden layer with the fewest number of neurons. The number of hidden layers affects how the encodings are calculated; it is not the same if the input is reduced in a single step or by gradually decreasing the number of neurons in the hidden layers. The experiments presented in this thesis evaluate the impact of autoencoder architecture on ML task accuracy and data reduction.

## 4.4 Multi-node Edge-cloud Computation Model

Unlike the single-node, the multi-node computation model involves several edge nodes, with each one responsible for processing some subset of data. Which node is responsible for which data differs for similarity and location-based models. Additionally, as this thesis is concerned with sensor data, the edge nodes perform data preprocessing, such as applying a sliding window technique. The multi-node edge-cloud computation model is depicted in Figure 4.4. It includes three main components: data preprocessing, data reduction, and Cloud ML, which are described in the following subsections.

### 4.4.1 Data Preprocessing

The edge-cloud ML process starts with data from IoT sensors being passed to the edge for preprocessing in contrast to traditional ML where the data are sent directly to the cloud. The preprocessing always includes normalization while the sliding window technique is optional, and its impact is evaluated in the experiments.

### Normalization

To avoid dominance of features with large values and to improve training convergence, the data are normalized using standardization ($z$-score). In place of standardization, *min-max* scaling could be used, but standardization was selected because of its ability to handle outliers. Each feature is rescaled to have zero mean and unit variance as given by equation:

$$\hat{x} = \frac{x - \mu}{\sigma} \tag{4.1}$$

where $x$ is the original feature value, $\mu$ and $\sigma$ are that feature mean and standard deviation, and $\hat{x}$ is the normalized value.

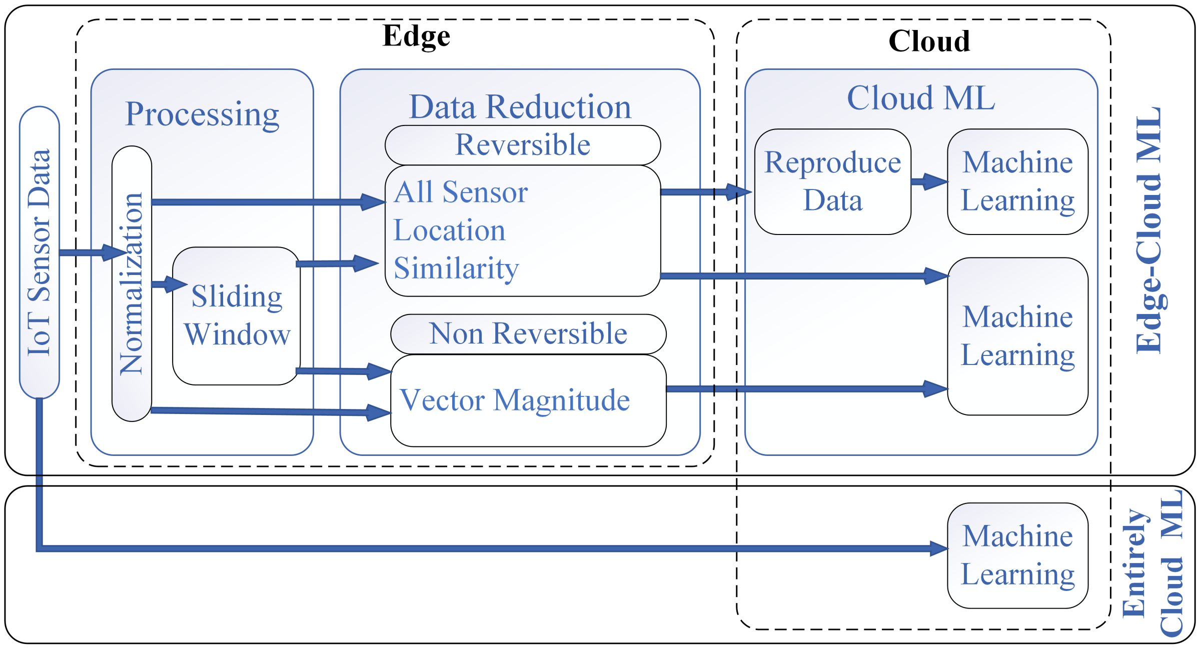

Figure 4.4: Multi-node edge-cloud computation model.

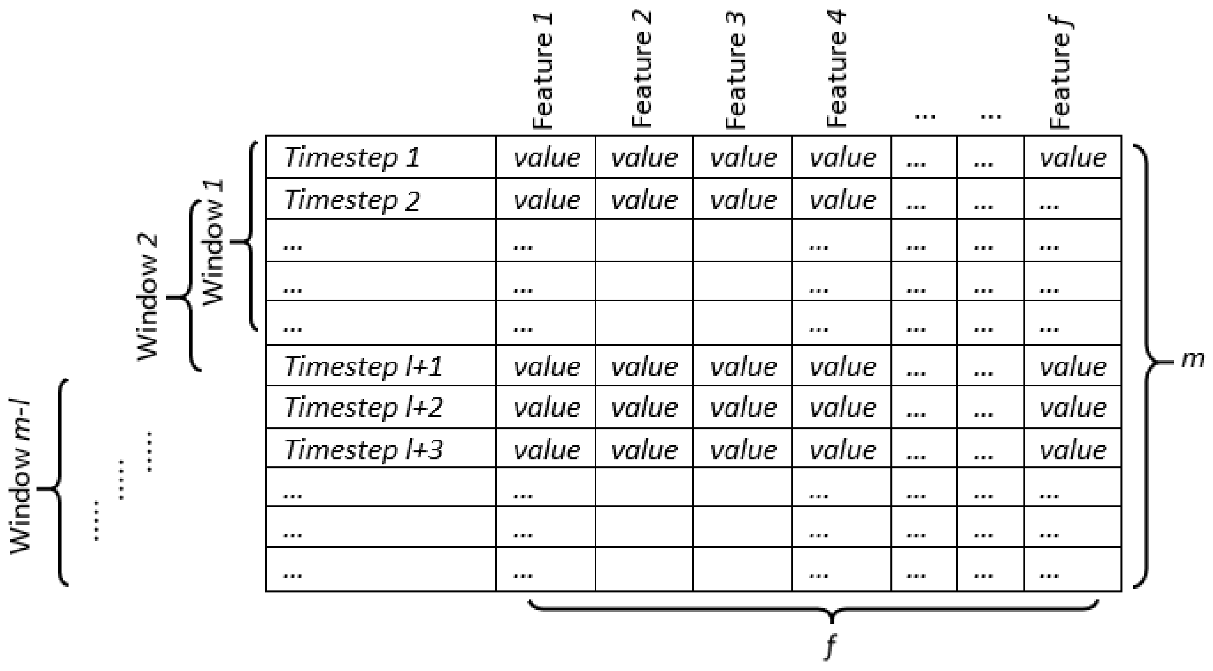

Figure 4.5: Sliding window strategy.

## Sliding Window

At this point, one data sample consists of several readings, potentially from different sensors and different locations, for the same time step $t$. For time series data, the window sliding technique is applied to help the model capture time-dependencies or to prepare data in the format needed by the ML algorithms [56]. If the data set has $m$ observations, with the sliding window of length $l$, the first sample consists of all readings for the first $l$ time steps; with $f$ number of features, this will result in $l \times f$ matrix for each sample. Next, the window slides for $k$ steps, and the second sample consists of the readings from time step $k$ to $k + l$. The window keeps sliding to create the remaining samples, as depicted in Figure 4.5. In this thesis, sliding step $k = 1$ is used as this results in a higher number of samples for training and allows the input to capture shifts in temporal patterns. Once the system is trained, different sliding steps can be applied depending on the specifics of the use case and data transfer constraints.

### 4.4.2 Data Reduction

Data reduction happens on the edge in order to reduce the quantity of data sent to the cloud. It can be carried out directly with normalized data or with samples created by the sliding window technique. Regardless of whether the sliding window technique is used or not, the data reduction approach is the same. Two types of approaches are considered: reversible and non-reversible.

#### Reversible

Reversible approaches are the approaches that reduce data with an ability to reproduce the original data from the reduced representations. With these approaches data reduction executes on the edge, reduced data are sent over the network, and on the cloud, ML can be performed directly on the reduced data, or the original data can be reproduced first. Here the focus is on autoencoders as, once the AE is trained, the encoder can reduce data on the edge, and decoder can restore the original data on the cloud. AE performance is compared to data reduction with PPCA [28]. When PCA is applied to reduce dimensions, original data can be reconstructed using the eigenvectors. For both AE and PCA, the accuracy of reconstructed data depends on the degree of dimensionality reductions.

As illustrated in Figure 4.4, for both reversible techniques, AE and PCA, three different scenarios are considered: all sensors, location-based, and similarity-based scenario. In the all sensors approach, all data are considered together, and data reduction is performed on the merged data from all sensors.

The location-based scenario considers data reduction based on a group of co-located sensors, as illustrated in Figure 4.6. For example, on location 1, there are four different sensors, and their data are sent to the edge node E1 because of its close proximity to those sensors. Similarly, location 2 sensors send data to E2, and so on. The idea is to keep the edge part of the processing as close as possible to the sources of data and reduce the distance data needs to travel before reduction. Data that arrives at one node are reduced together on that node;

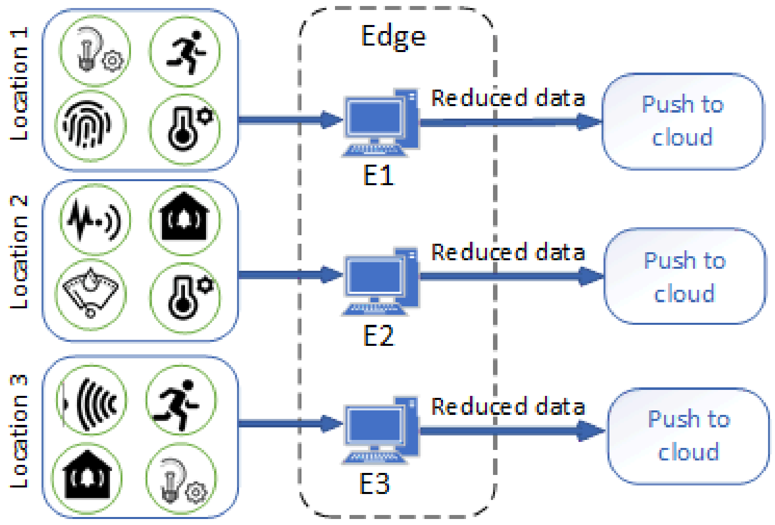

Figure 4.6: Location-based data reduction.

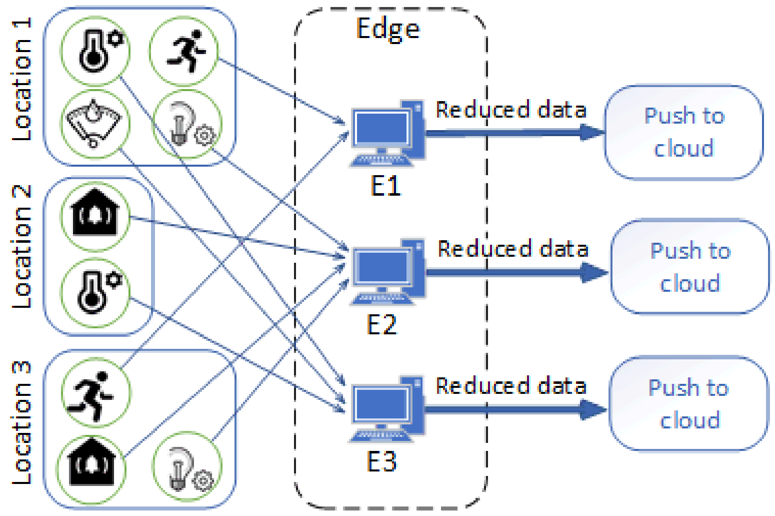

Figure 4.7: Similarity-based data reduction.

consequently, there is one AE for each of the edge nodes.

The similarity-based scenario illustrated in Figure 4.7 groups sensors based on their similarity. For example, all gyroscopes could represent one group and all accelerometers another one. This will result in more homogeneous data groups, but it can also increase the distance of sensors from the edge nodes. As with the location-based scenario, there is one AE for each of the edge nodes.

**Non-reversible**

Non-reversible approaches include those without the way of reproducing the original data after the data have been reduced. Here we considered a single approach, vector magnitude, suitable for sensors that measure values in multi-dimensional cartesian space such as accelerometers and gyroscopes. The dimensionality is reduced as follows:

$$d = \sqrt{x^2 + y^2 + z^2} \tag{4.2}$$

Here $x$, $y$, $z$ are the measurements in cartesian coordinates and $d$ is the vector magnitude. Consequently, vector magnitude reduces dimensions in 3:1 ratio.

### 4.4.3 Cloud ML

The reduced data are sent from the edge to the cloud for further ML processing. As illustrated in Figure 4.4, there are two possible ways to carry out the ML task. The first option is to reproduce original data and use these reproduced data for the ML task. This is possible only if the reversible technique was used for data reduction. The section option is to carry out the ML task directly on the reduced data that works for both reversible and non-reversible reduction techniques. As the reproduced data have a larger number of features, training ML model for such data is more computationally expensive than training ML model for the reduced number of features.

# Chapter 5

# Evaluation Methodology

As this thesis considers single-node and multi-node computation models, the evaluation methodology is described separately for the two models. The two evaluations are designed for different types of data sets. The single-node model considers already preprocessed data set where temporal features are extracted from data, whereas the multi-node model considers time-series data arriving directly from different sensors. Moreover, the multi-node model examines different strategies for assigning work to nodes.

## 5.1 Single-node Edge-cloud Computation Model

This section first describes the machine learning task and data set used for evaluating the single-node model. Next, experiments designed for the single-node approach are described.

### 5.1.1 Machine Learning Task and Data Set

The presented approach has been evaluated on the task of human activity recognition from smartphone data. Because smartphones are equipped with a variety of sensors such as accelerometers, gyroscopes, and proximity sensors, and support wireless communication protocols such as Wi-Fi and Bluetooth, they are capable of collecting and transmitting large quantities of data related to human activities. Pervasiveness makes smartphones a convenient and affordable solution for the unobtrusive monitoring of human activities [57].

This thesis uses publicly available human activity data set [57, 58]. Anguita *et al.* created this data set from accelerometer and gyroscope readings at a sampling rate of 50 Hz [57]. Data have been preprocessed with a fixed-length sliding window by calculating various features such as average and standard deviation for each window. In the available data set, there are 561 numeric features. Additionally, each sample in the data set is labelled as walking, walking down, walking up, sitting, standing, or laying. Thus, this is a classification problem. As this data set contains a relatively large number of features (561), it is suitable for data reduction on the edge nodes using deep learning.

The data set has 10301 observations; 70% of the data were used for training, and remaining 30% for testing. The data set was already scaled to the [-1,1] range, and thus normalization was not performed. Note that in the proposed approach, the training is carried out on a single centralized system, and then the encoder part is deployed onto the edge node and the decoder part on the cloud.

### 5.1.2 Experiments

Three types of experiments were performed corresponding to the three scenarios from Figure 4.3:

- Classification with *encoded* data: Data are reduced (encoded) and the encoded data are used for the classification.
- Classification with *decoded* data: Data are first reduced (encoded), then restored (decoded), and the decoded data are used for the classification.
- Classification with *original* data: Data are used unchanged for the classification. This scenario is included for comparison purposes.

The AE's hidden layer with the fewest neurons determines how much data are reduced. To explore different degrees of reduction, the following architectures were considered:

- 561-265-561
- 561-265-128-265-561
- 561-265-128-64-128-265-561

Here each number represents a number of neurons in a layer starting from the input layer to the output layer. All architectures are symmetrical: encoder parts reduce the number of features (e.g., 561-256-128), and the corresponding decoders restore original data (e.g., 128-256-561). Consequently, the three architectures reduce data to 265, 128, and 64 features, respectively. In the case of a larger dimensionality reduction (e.g., 128 or 64), additional layers are introduced. Experiments with a large direct reduction were also performed, but gradual reduction achieved better accuracy.

Overall classification network hyperparameters were as follows:

- Optimizer: Adam
- Epochs: 100
- Loss: MSE
- Activation: ReLu + Softmax

Here Adam [59] is a first-order stochastic objective functions with gradient-based optimizer which includes lover-order of moments. ReLU [60] is used as an activation function in hidden layers. The minimax rates of convergence of ReLU is up to *logn*-factors under a general composition assumption on the regression function. With Softmax [61] function as their classification function.

Table 5.1 shows architectures for classification neural networks used on the cloud. The number of input neurons is equal to the number of features after the reduction, while the number

Table 5.1: Classification neural network architecture.

| | **Features after reduction** | | | |
|---|---|---|---|---|
| | **561** | **256** | **128** | **64** |
| Encoded | - | 256-128-32-6 | 128-32-6 | 64-16-6 |
| Decoded | - | 561-265-128-64-16-6 | 561-128-64-6 | 561-256-128-64-32-6 |
| Original | 561-128-32-6 | - | - | - |

of output neurons is always six as there are six classes. It can be seen that three degrees of reduction were considered 256, 128, and 64. There is also a column with 561 features, and this is for the original data without any reduction. Experiments were performed with other architectures, but those presented in Table 5.1 achieved better results.

## 5.2 Multi-node Edge-cloud Computation Model

This model involves multiple edge nodes, and consequently, different data reduction approaches on the edge nodes. Moreover, different data preparation strategies are considered as well as varying degrees of data reduction. The evaluation process is described in the order of the main components of the architecture: data preprocessing (including the data set used for the evaluation), data reduction, and cloud ML

### 5.2.1 Data Preprocessing

While the single-node evaluation considered already preprocessed data set, multi-node evaluation deals with unprocessed time series data from the sensors. This allows for the evaluation of different preprocessing techniques and their impact on the machine learning task accuracy. Specifically, the multi-node model was evaluated on the HAR tasks with the MHEALTH dataset [14]. This thesis takes a sensor-based approach where activity types such as walking, running, or sitting, are determined with the help of sensors. The MHEALTH dataset contains recordings of body motion for ten individuals while performing different activities. The recordings were acquired by three types of sensors, accelerometer, gyroscope, and magnetometer, with each one obtaining three readings corresponding to three axis. All three sensors are placed on the left ankle and the right wrist while on the chest, there is only the accelerometer. This makes a total of 21 features: 7 sensors $\times$ 3 axis. The labels are 12 physical activities (standing, walking, etc.) recorded with 10 individual subjects, the sampling rate is 50Hz, and the total number of samples is 343195.

Data from all individuals are merged together and used as such throughout experiments. For the training and the test, data were split 80:20, respectively. As illustrated in Figure 4.4, after data are normalized, the preprocessing can continue with or without applying the sliding window. To evaluate the impact of the window size on activity detection accuracy and on obtainable reduction rate, three window sizes are considered: 25, 50, and 100 readings. With the 50 Hz sampling rate, these sizes correspond to 0.5, 1, and 2 seconds.

### 5.2.2 Data Reduction

The two categories of data reduction on the edge are considered: reversible and non-reversible.

#### Reversible

Reversible data reduction with and without sliding window is applied with three scenarios: all sensors, location-based, and similarity-based. Table 5.2 shows the number of features before and after data reduction for reversible technique with window size 100 for the three scenarios. In the table and remainder of the thesis, the term *Direct* is used to refer to the approach without the sliding window.

The table gives the number of features before and after reduction for 66%, 70%, 80%, 90%, and 95% reduction. The starting reduction of 66% was selected to match the maximal reduction of the vector magnitude approach 3:1. For all sensors, there is no edge location, as everything happens on a single node, and this is the single-node strategy. For location-based and similarity-based approaches, edge location identifies a node where data are aggregated. For the location-based approach, three nodes are corresponding to sensors located on the arm, leg, and chest. For the similarity-based approach, the nodes correspond to the type of sensor: accelerometer, gyroscope, and magnetometer. For example, L2 in location-based techniques aggregates readings from three sensors; therefore, the number of original features for the direct approach is 9 (3 sensors with 3 axis readings each), as shown in Table 5.2. For scenarios with windows, the number of features is the window length $\times$ the number of features; for example, for location-based approach and L2, the number of features is $9 \times 100$.

It is important to note that the direct option is only used with 66% data reduction rate. At that value, the number of features is already very low, for example, 7 for all sensors, and further reduction is considered undesirable.

The number of features in the approaches with the sliding window is much higher than in direct approaches, but with sliding windows, buffering happens on the edge and only the compressed data are sent to the cloud. Moreover, it is expected that the sliding window technique will achieve better accuracy because of its past successes in HAR [50].

All of the data reduction variants shown in Table 5.2 are carried out with AE and with PCA. Regardless if PCA or AE are used, data reduction rates remain the same as presented in Table 5.2. For AE, a reduction degree is controlled by setting the number of neurons in the bottleneck layer, where for PCA, it is controlled by the number of selected principal components. For the location-based and similarity-based approaches, there is one AE for each edge node responsible for reducing data that are arriving on that node. Similarly, PCA works independently on each node.

Table 5.2 shows data only for sliding window 100 but the same strategy was applied for sliding windows 50 and 25: number of features for sliding windows 50 and 25 are provided in Appendix A, tables A.1 and A.2.

Overall, there are 90 data reduction experiments with sliding windows: 3 scenarios $\times$ 3 window sizes $\times$ 5 reduction rates $\times$ 2 algorithms (AE, PCA). Additionally, for direct reduction,

Table 5.2: Number of features before and after reduction for reversible approaches (window size 100).

| **Scenario** | **Edge Location** | **Original Features** | **66% Reduction** | **70% Reduction** | **80% Reduction** | **90% Reduction** | **95% Reduction** |
|---|---|---|---|---|---|---|---|
| **All Sensors** | | | | | | | |
| Direct | - | 21 | 7 | - | - | - | - |
| S.Window | - | $21 \times 100$ | 693 | 630 | 420 | 210 | 105 |
| **Location Based** | | | | | | | |
| Direct | L1 | 3 | 1 | - | - | - | - |
| | L2 | 9 | 3 | - | - | - | - |
| | L3 | 9 | 3 | - | - | - | - |
| S.Window | L1 | 3×100 | 99 | 90 | 60 | 30 | 15 |
| | L2 | 9×100 | 297 | 270 | 180 | 90 | 45 |
| | L3 | 9×100 | 297 | 270 | 180 | 90 | 45 |
| **Similarity Based** | | | | | | | |
| Direct | S1 | 9 | 3 | - | - | - | - |
| | S2 | 6 | 2 | - | - | - | - |
| | S3 | 6 | 2 | - | - | - | - |
| S.Window | S1 | 9×100 | 297 | 270 | 180 | 90 | 45 |
| | S2 | 6×100 | 198 | 180 | 120 | 60 | 30 |
| | S3 | 6×100 | 198 | 180 | 120 | 60 | 30 |

there are 6 experiments: 3 scenarios × 2 algorithms. This makes a total of 96 data reduction experiments.

#### Non-Reversible

The only non-reversible approach considered is the vector magnitude. With three-axis data as those recorded with an accelerometer, gyroscope, and magnetometer, this results in 3:1 reduction. As with reversible approaches, the reduction is applied with and without the sliding window. Without the window, vector magnitude for all data reduces 21 features to seven. For sliding windows 25, 50, and 100, features are reduced from 525, 1050, and 2100 to 175, 350, and 700. This makes a total of 4 experiments: 3 for each window size + direct reduction (no sliding window).

### 5.2.3 Cloud ML

After the data are reduced on the edge, they are sent to the cloud for the final processing. In the HAR task, this final step is classification: recognizing the type of activity. While on the edge there are ML models on each of the nodes taking care of processing data arriving at that node, on the cloud, there is a single model merging data from all edge nodes. Here, as illustrated in Figure 4.4, three approaches have been considered for carrying out this task:

#### ML with Reduced Data

Reduced data arriving from all edge nodes are used as such directly for classification. This approach is applicable for both reversible and non-reversible reduction techniques. For the HAR task, a feed-forward neural network (FFNN) was used for the final classification. The number of output nodes is 12 (12 activities), and the number of input nodes in the FFNN corresponds to the number of features after the data reduction and reduces with increased data reduction rates. A different number of hidden layers and neurons was used depending on the number of input features: for the number of input features more than 350, FFNN had three hidden layers, and for 350 or less, it had two hidden layers. The number of nodes in the hidden layers gradually decreased, for example, 420-128-32-12 for 420 input features. Experiments were performed with different numbers of neurons in the hidden layers, but this strategy showed high accuracy.

**ML with Reproduced Data:**

This approach consists of two steps, reproducing the original data from their reduced representation and then classifying the reproduced data. Clearly, this is only applicable when reversible techniques are used to reduce data on the edge. As the number of features here matches the number of features in the original data set, this approach requires more complex models, and it is more computationally expensive than ML with reduced data. However, as the original data are reproduced, these reproduced data can be used for other tasks. As with ML with reduced data, classification is carried out with the FFNN, and the same strategy is applied for choosing the number of layers and neurons in the hidden layers. As there is no reduction, the same strategy resulted in a higher number of layers and neurons.

**Entirely Cloud-based ML**

This is not an edge-cloud architecture, but a traditional cloud-based technique where everything is sent to the cloud for ML. It is considered in the evaluation solely for comparison reasons. Again, FFNN is used for classification, and the same strategy was applied for determining the number of layers as with edge-cloud approaches.

As discussed in section 5.2.2 evaluation considers 96 experiments that were performed for reversible techniques. With two approaches on the cloud (with reduced and with reproduced data), this makes a total of 192 experiments with reversible techniques. There are 4 experiments for non-reversible approaches (section 5.2.2) and 4 (one direct and 3 sliding windows) for entirely cloud-based ML. This makes for a total of 200 experiments.

# Chapter 6

# Results and Discussion

This chapter discusses the result of the evaluation processes presented in Chapter 5. As the evaluation has been performed for the two models, single-node and multi-node, the results are discussed separately. This chapter first presents the results for the single-node computation model and discusses finding. Next, multi-node computation model results are presented, and the chapter concludes with a discussion.

All of the models were implemented in Python with PyTorch deep learning library. The experiments were performed on a computer with Ubuntu OS, AMD Ryzen 4.20 GHz processor, 128 GB DIMM RAM, and four NVIDIA GeForce RTX 2080 Ti 11GB graphics cards. Training machine learning models is computationally intensive, and with a large number of experiments, the process is long; therefore, GPU acceleration was used. However, once the model is trained, it was tested with CPU resources.

## 6.1 Single-node Computation Model

Two aspects of the system were evaluated: the impact of data reduction on the ML task accuracy and the degree of data reduction. With Big Data, a small drop in accuracy can be accepted if it is accompanied by gains such as reduction of network traffic and latencies.

As a benchmark for data reduction, a classification with a complete data set was carried out

Table 6.1: Classification Accuracy.

| Accuracy | MSE | MAE |
|---|---|---|
| 0.9545 | 0.01311 | 0.01752 |

first (Scenario 3 from Section 5.1); results are presented in Table 6.1. The achieved accuracy is high: over 95% of samples are classified correctly. Moreover, MSE and MAE are also low.

Next, Scenario 1 and 2 experiments were conducted with the three autoencoder (AE) architectures presented in subsection 5.1.2: Figure 6.1 shows the results. Values on the horizontal axis 256, 128, and 64 show the number of features after data reduction. Scenario 1 is indicated as *encoded* and Scenario 2 as *decoded*. In the figure, *'original'* refers to the classification with original data (Scenario 3) and, therefore, has the same values regardless of the horizontal axis reduction group. It can be observed that accuracy decreases as the number of features is reduced. Nevertheless, for 256 features, although the number of features is reduced from 561 to 256, there is hardly any change in accuracy as it just changes from 95.45% to 95.31%. Similarly, the reduction to 128 features only reduces accuracy to 94.46%.

However, there is a more significant change in accuracy when going to 64 features: accuracy was 90% and 87% for encoded and decoded data, respectively. However, note that this represents 88% reduction in data size. With 128 features and with 64 features, classification with encoded data achieves better accuracy than classification with decoded data; the difference is especially large in the case of 64 features.

Figure 6.2 shows the results for the same experiments but using PCA in place of an autoencoder. Still, data reduction to 256, 128, and 64 is considered. As with autoencoders (Figure 6.1), with PCA accuracy decreases as features are reduced. For encoded data and 256 features, accuracy is better than the accuracy with all features. This can be explained by 'curse of dimensionality', the negative effect of a large number of features on the accuracy of ML algorithms [3].

Reduction using an autoencoder and PCA is compared in Figure 6.3; an AE with encoded

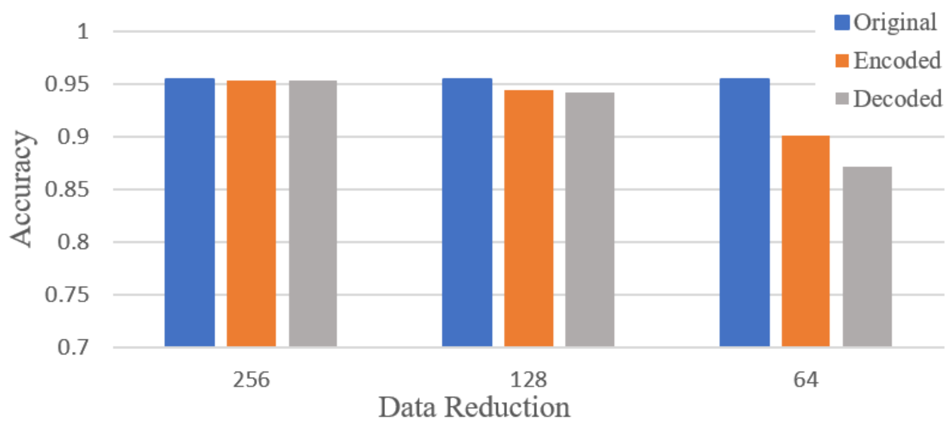

Figure 6.1: Classification accuracy with autoencoder data reduction.

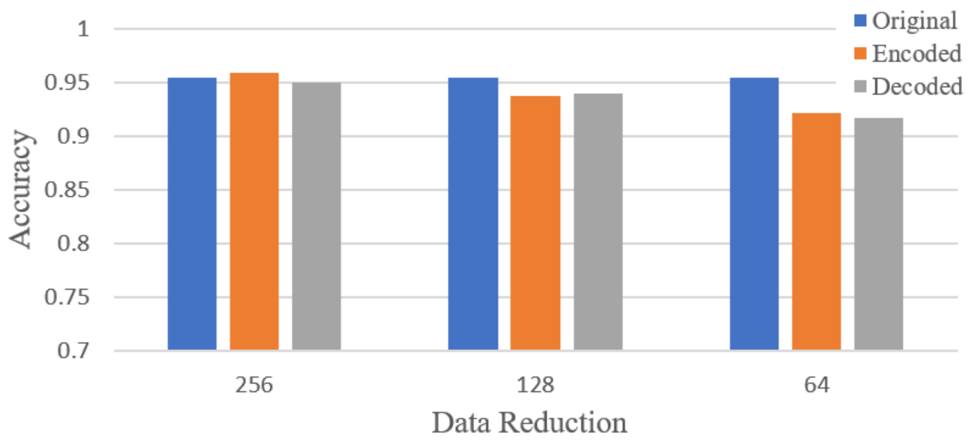

Figure 6.2: Classification accuracy with PCA data reduction.

and decoded data and PCA with encoded and decoded data are considered. When data are reduced to 256 features, all four approaches achieve similar accuracy, with PCA encoded data achieving slightly better accuracy. When data are reduced to 128 features, autoencoders outperform PCA with the encoded version performing somewhat better then decoded. Reducing features to 64 results in a significant drop in accuracy with both PCA and AE. Nevertheless, even with an aggressive reduction to 64 features, which is only around 13% of the original 561 features, the accuracy of AE with encoded data is still around 87%.

Because the main objective of edge-cloud IoT data analytics is to reduce network traffic and latencies, it is important to examine how much the proposed approach reduces data size. Figure 6.4 shows the size of the original data compared to reduction to 256, 128, and 64 features. It

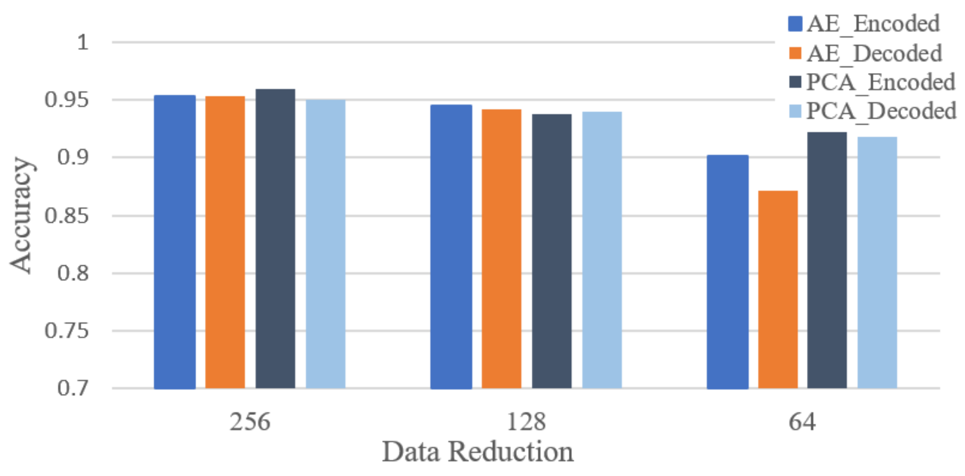

Figure 6.3: Autoencoder and PCA comparison.

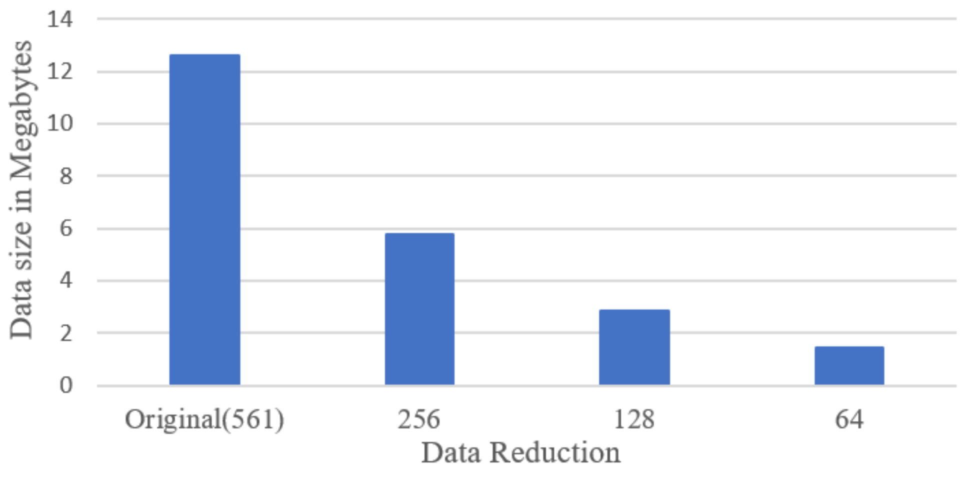

Figure 6.4: Data size for different feature reduction.

can be seen that data reduce from 11.2 MB for original data to 5.754 MB for 256 features, 2.877 MB for 128, and 1.4386 MB for 64 features. Consequently, the data sent to the cloud are significantly reduced, which is especially important in the case of large data quantities such as those in IoT.

The experiments presented here show that by using autoencoders it is possible to reduce the number of features from 561 to 256, which represents a reduction of over 50%, without significantly impacting the accuracy of the human activity recognition task. Although PCA is a linear feature reduction technique, it achieved similar results to the autoencoder, which is a non-linear technique. Consequently, it cannot be concluded that autoencoders always outperform PCA. Nevertheless, both autoencoders and PCA were successful in reducing data without significantly impacting the ML task accuracy, and thus both can be used for edge-cloud analytics.

## 6.2 Multi Edge-node Model

This section presents results from the evaluation of the multi-node computation model.

### 6.2.1 Result

First, to enable comparisons, the accuracy of traditional ML without any data reduction is presented in Table 6.2. All sliding window experiments outperform a non-window, direct approach. Next, Table 6.3 shows classification accuracy for reversible techniques with 66% data reduction and sliding window size 100. It can be observed that similar to traditional ML (Table 6.2), the sliding window technique achieves better accuracy than direct approaches, regardless of the algorithm (AE or PCA) or scenario (all sensors, location-based, or similarity-based). However, there is no big difference between AE and PCA or between reduced and reproduced approaches. Although data reduction was 66%, the accuracy is very close to the accuracy of the traditional cloud-based ML. Precision and recall were also calculated; they showed similar patterns to the accuracy observed in Table 6.3. The same experiments were carried out for 70%, 80%, 90%, and 95% reduction and their results are available in Appendix A, Tables A.3, A.4, A.5, and A.6, respectively.

Table 6.2: Classification accuracy: entirely cloud ML.

| | **Direct** | **Window 100** | **Window 50** | **Window 25** |
|---|---|---|---|---|
| **Accuracy** | 98.94% | 100% | 100% | 99.99% |

Table 6.3: Classification accuracy for reduced and reproduced data: 66% data reduction.

| Category | AE | | | | | | PCA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reduced | | | Reproduced | | | Reduced | | | Reproduced | | |
| | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall |
| **All Sensors** | | | | | | | | | | | | |
| Direct | 98.27% | 0.98 | 0.98 | 99.24% | 0.99 | 0.99 | 98% | 0.98 | 0.98 | 98.17% | 0.98 | 0.98 |
| S.Window 100 | 100% | 1 | 1 | 100% | 1 | 1 | 100% | 1 | 1 | 100% | 1 | 1 |
| S.Window 50 | 99.92% | 1 | 1 | 99.95% | 1 | 1 | 99.9% | 1 | 1 | 99.9% | 1 | 1 |
| S.Window 25 | 99.7% | 1 | 1 | 99.9% | 1 | 1 | 99.6% | 1 | 1 | 99.84% | 1 | 1 |
| **Location Based** | | | | | | | | | | | | |
| Direct | 98.13% | 0.98 | 0.98 | 98.44% | 0.98 | 0.98 | 99.4% | 0.99 | 0.99 | 99.38% | 0.99 | 0.99 |
| S.Window 100 | 99.89% | 1 | 1 | 99.92% | 1 | 1 | 99.89% | 1 | 1 | 99.86% | 1 | 1 |
| S.Window 50 | 99.9% | 1 | 1 | 99.94% | 1 | 1 | 99.9% | 1 | 1 | 99.9% | 1 | 1 |
| S.Window 25 | 99.58% | 1 | 1 | 99.89% | 1 | 1 | 99.39% | 0.99 | 0.99 | 99.86% | 1 | 1 |
| **Similarities based** | | | | | | | | | | | | |
| Direct | 98.74% | 0.98 | 0.98 | 99.24% | 0.99 | 0.99 | 99.32% | 0.99 | 0.99 | 99.34% | 0.99 | 0.99 |
| S.Window 100 | 99.90% | 1 | 1 | 99.92% | 1 | 1 | 99.86% | 1 | 1 | 99.85% | 1 | 1 |
| S.Window 50 | 99.91% | 1 | 1 | 99.96% | 1 | 1 | 99.9% | 0.99 | 0.99 | 99.91% | 1 | 1 |
| S.Window 25 | 99.62% | 1 | 1 | 99.89% | 1 | 1 | 99.35% | 0.99 | 0.99 | 99.86% | 1 | 1 |

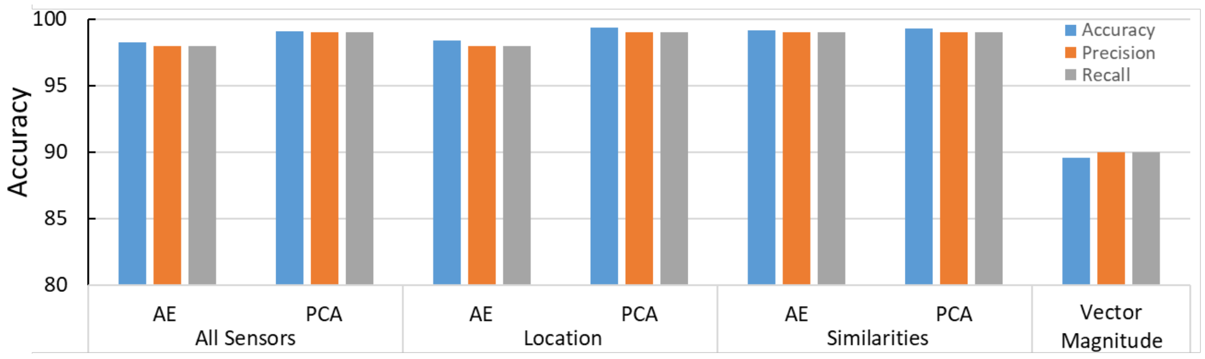

Figure 6.5: Classification accuracy: direct (no Window), 66% reduction, reversible and non-reversible approaches.

Like Table 6.3, Figure 6.5 is concerned with 66% reduction, but it includes both, reversible and non-reversible approaches, in contrast to Table 6.3 which considers only reversible approaches. All reversible approaches achieve much higher accuracy than the non-reversible, vector magnitude approach.

Table 6.3 and Figure 6.5 analyze classification accuracy for 66% data reduction and Table 6.3 considers solely sliding window 100.

To analyze the impact of data reduction rates on the accuracy, Figure 6.6 illustrates accuracy changes for sliding window 100 with respect to reduction rate for each of the three approaches: all sensors, location-based, and similarity-based.

It can be observed that accuracy slowly decreases as reduction increases from 66% to 90%, and then experiences sharper drop with 95% reduction rate. Overall, AE show slightly higher accuracy than PCA, and classification with reproduced data is marginally better than with reduced data. Nevertheless, even with a 95% reduction rate, the accuracy drops less than 0.25% in comparison to traditional ML.

Figure 6.7 shows the same evaluation as Figure 6.6, but for window size 50. While for window size 100, the accuracy remained relatively stable for reductions of 66% to 80%, with sliding window 50, the accuracy drops as reduction rate changes. Nevertheless, the accuracy for 95% reduction is similar for windows 100 and 50; however, window 50 is more desirable as it shortens FFNN training time and reduces required data accumulation on the edge. As

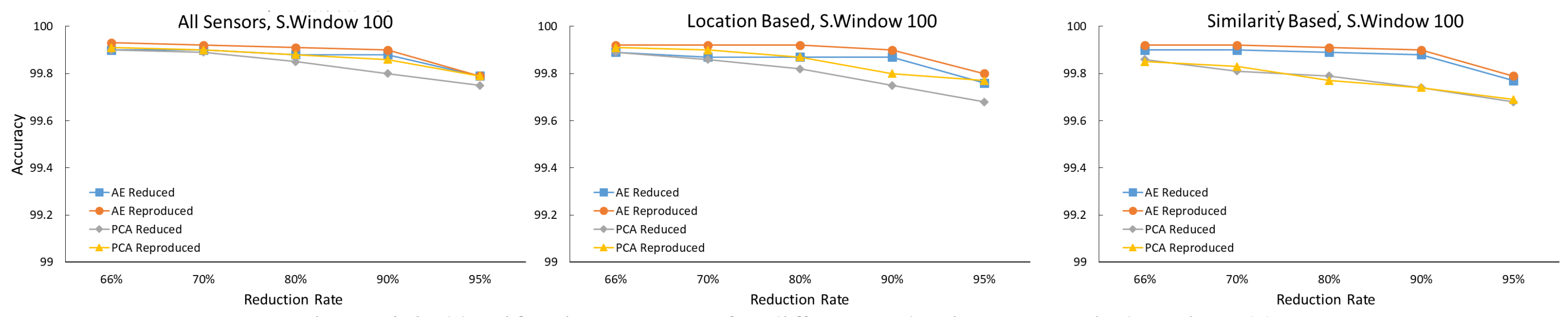

Figure 6.6: Classification accuracy for different reduction rates: window size 100.

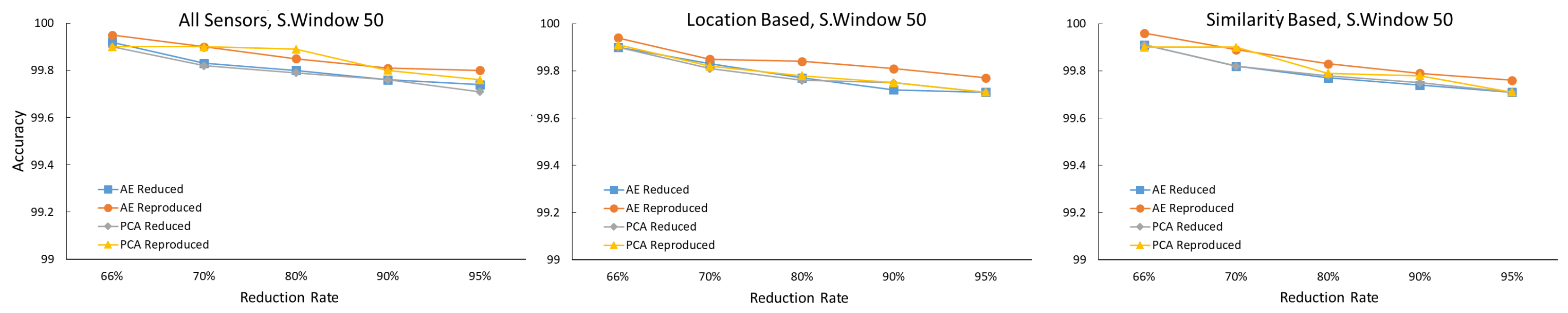

Figure 6.7: Classification accuracy for different reduction rates: window size 50.

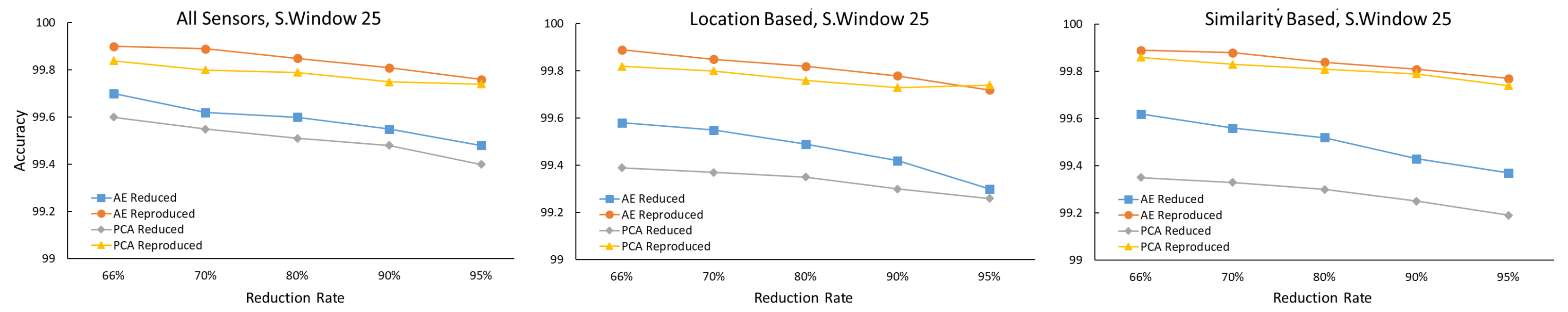

Figure 6.8: Classification accuracy for different reduction rates: window size 25.

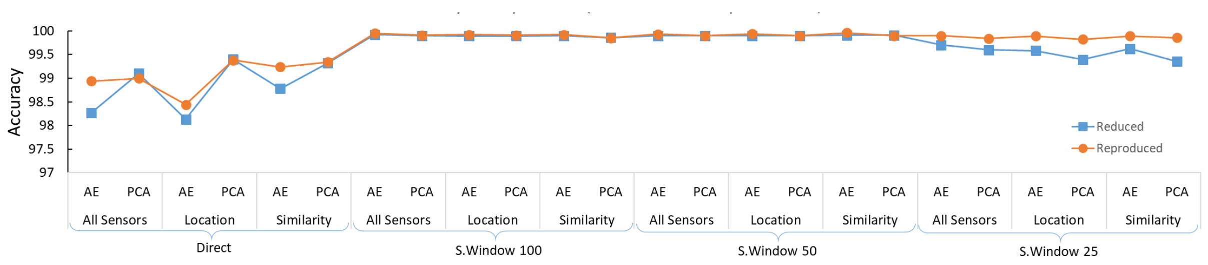

Figure 6.9: Classification accuracy: direct and sliding window approaches, 66% reduction rate.

with window size 100, with size 50, AE with reproduced data is slightly better than the other approaches.

Next, Figure 6.8 shows accuracy for window size 25. Whereas with windows sizes 100 and 50, classification with reduced and reproduced data achieved similar accuracy, with window size 25, approaches with reduced data show lower accuracy than with reproduced data for all reduction rates and both algorithms, PCA and AE. As with other window sizes, AE with reproduced data achieved the highest accuracy for almost all reduction rates. Comparing sliding window 25 and 100 results for AE with reproduced data, sliding window 100 shows slightly better accuracy, but window 25 may be more desirable because of its shorter training time.

Figures 6.6, 6.7, and 6.8 compare the accuracy of reversible approaches for different window sizes, but it is also important to compare sliding window approaches with the direct approaches: Figure 6.9 shows this comparison. As direct approaches are only considered for a 66% reduction rate, this figure compares them to sliding window approaches with the same reduction rate. All sliding window approaches, regardless of the size of the window, outperform direct approaches further confirming the benefit of using sliding windows. As with figures 6.6, 6.7, and 6.8, Figure 6.9 show that windows 100 and 50 achieve similar accuracy while for window size 25, accuracy for reduced data decreases.

While Figure 6.9 shows the classification accuracy for 66% reduction rate, Figures 6.10-6.13 depict the same for data reduction rates of 70%, 80%, 90%, and 95%, respectively. Figure 6.9 includes direct approaches while the others do not because, as previously mentioned, the direct reduction was only carried out for 66% reduction rate. The direct reduction was only

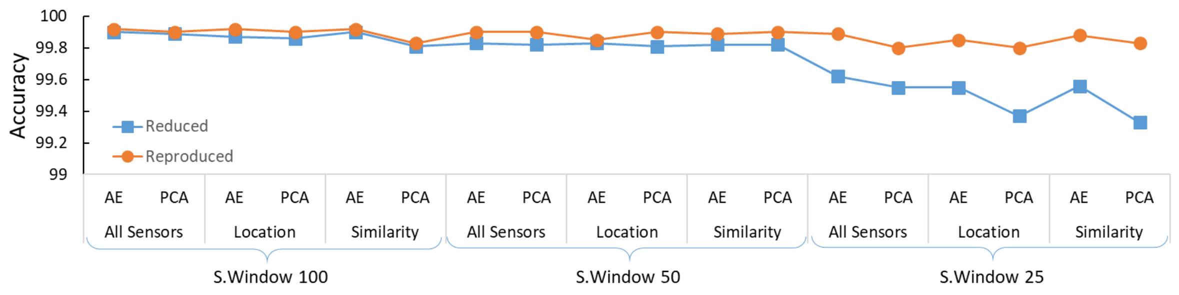

Figure 6.10: Classification accuracy: sliding window approaches, 70% reduction rate.

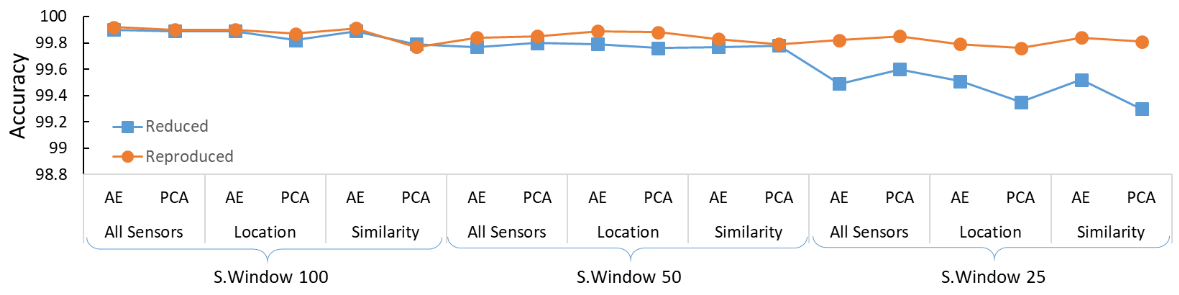

Figure 6.11: Classification accuracy: sliding window approaches, 80% reduction rate.

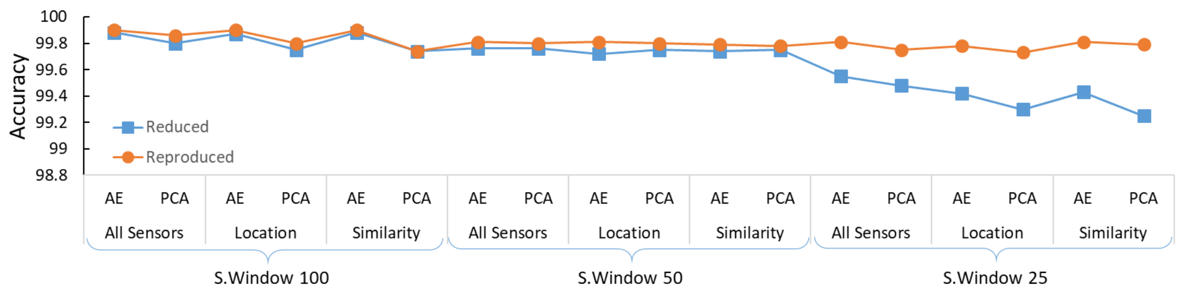

Figure 6.12: Classification accuracy: sliding window approaches, 90% reduction rate.

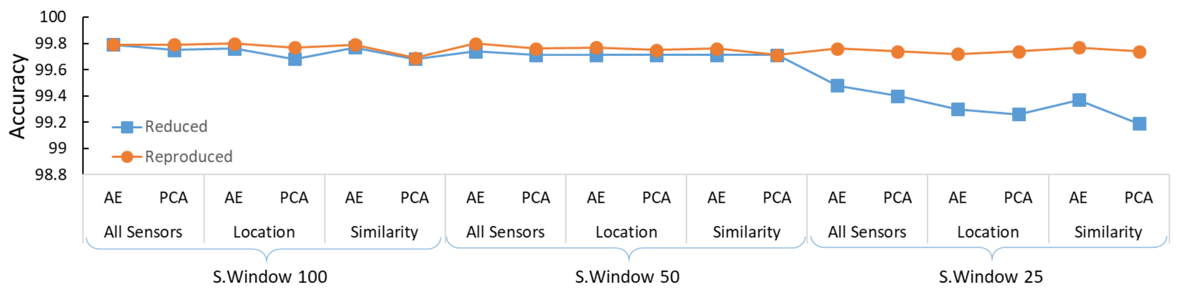

Figure 6.13: Classification accuracy: sliding window approaches, 95% reduction rate.

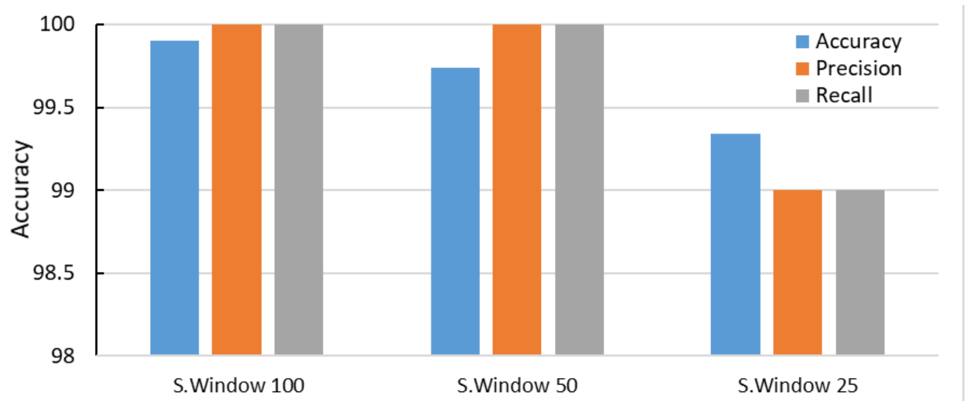

Figure 6.14: Classification accuracy: vector magnitude with sliding windows.

performed for 66% reduction rate as even with that rate, the number of reduced features was low; for example, in the location-based approach, for L2 location, the number of reduced features was three as illustrated in Table 5.2.

All four Figures 6.10-6.13 show a similar pattern to the one observed in Figure 6.9 for sliding window approaches. For sliding window 100 and 50, the accuracy is similar for both AE and PCA, and for both, reduced and reproduced approaches. However, for sliding window 25, there is a noticeable difference between reduced and reproduced data: classification with reproduced data is achieving higher accuracy. Moreover, the accuracy of classification with reproduced data and sliding window 25 is similar to the accuracy archived with sliding windows 100 and 50. This same pattern is observed for all reduction rates as illustrated in Figures 6.9-6.13. As the input to the classification network is the number of features $\times$ the window size, the combination of a low number of features with a short window may be resulting in inadequate input representation and causing a drop in accuracy.

Finally, the accuracy of non-reversible approaches for different window sizes is depicted in Figure 6.14. As with reversible approaches, window sizes 100 and 50 achieve similar results while accuracy for window size 25 is lower. Comparing the accuracy of the non-reversible approach with windows (Figure 6.14) and the same approach without windows (direct) (Figure 6.5), the accuracy is increased almost 10% by adding sliding windows. With sliding window, the non-reversible vector magnitude approach reaches accuracy close to reversible techniques; nevertheless, vector magnitude is computationally much more simple than the other approaches.

## 6.3 Network Traffic

So far, the analysis has considered data reduction in terms of number of features. However, when the system is deployed, the actual traffic will not exactly follow the feature reduction rates due to network overheads. To examine network traffic, the describe edge-cloud system has been simulated. The direct approach only needs a single edge node while location and similarity-based scenarios require three nodes, one for each location/similarity group, as discussed in Section 4.4.2. Each edge node is simulated by one virtual machine with 2GB of memory and single core CPU. The cloud was emulated by a window server with 6GB of memory and CPU with octa-core processor. The communication between the edge nodes and the server was established with the TCP-IP protocol and consequently, the consumed bandwidth was measured over sockets. Multithreaded socket server ensures that multiple edge nodes can communicate with the server concurrently. Encoder parts of the trained autoencoders are located on the edge nodes and decoder parts as well as the classification models are deployed on the server. The change in the computation capacity of the edge nodes would not change the network traffic, but it must be sufficient to carry out data encoding. The minimum computing resources required on the edge depend on the size of the encoder network which is affected by the sliding window size.

Table 6.4 compares the network traffic without feature reduction with the traffic in presence of 66% feature reduction for each of the scenarios. It also includes traffic reduction percentages indicating how much traffic was reduced in respect to the same scenario without reduction. For example, traffic reduction 49.71% for sliding window 100, for all sensors, indicates change from 7129 MB for no reduction to 3585MB for all sensors, for sliding window 100. It can be observed that for each scenario, network traffic remains similar for all three scenarios, all sen-

Table 6.4: Network traffic: no reduction and 66% data Reduction.

| Scenarios | No Reduc. | Reduced All Sensors | Reduced Location Based | Reduced Similarity Based |
|---|---|---|---|---|
| **Consumed Bandwidth MB** | | | | |
| Direct | 80.30 | 32.62 | 33.14 | 32.95 |
| S.Window 100 | 7129.01 | 3585.40 | 3646.89 | 3612.23 |
| S.Window 50 | 3548.76 | 1669.28 | 1690.51 | 1680.22 |
| S.Window 25 | 1842.30 | 838.88 | 841.28 | 840.78 |
| **Traffic Reduction %** | | | | |
| Direct | - | 59.38 | 58.73 | 58.97 |
| S.Window 100 | - | 49.70 | 48.84 | 49.33 |
| S.Window 50 | - | 52.96 | 52.36 | 52.65 |
| S.Window 25 | - | 53.46 | 54.33 | 54.36 |

sors, location-based, and similarity-based. However, for both, reduced and non-reduced data, network traffic is much higher for scenarios with sliding windows than for direct approaches. The reason for this is that the sliding step size $k = 1$ results in readings belonging to different sliding windows and thus the same readings are included in the transfer multiple times. This can be avoided by using the siding step equal to the length of the sliding window if the application allows for window length delays.

Fig. 6.15 compares network traffic reduction for different data reduction approaches and scenarios, for 66% feature reduction. Similar conclusions can be drawn as with Table 6.4: scenarios (all sensors, location, and similarity) do not have major impact on traffic reduction. While feature reduction was 66% for all approaches and scenarios, network traffic reduction varied among approaches: the direct approach resulted in the highest reduction (around 59%) and, as the window size increased, the reduction rate decreased. These variations are caused by changes in network overheads as the quantity of data sent altogether changes. Although window size 25 has an advantage of reduced network traffic in comparison to window sizes 50 and 100, its accuracy is lower, indicating the need to compromise between the two.

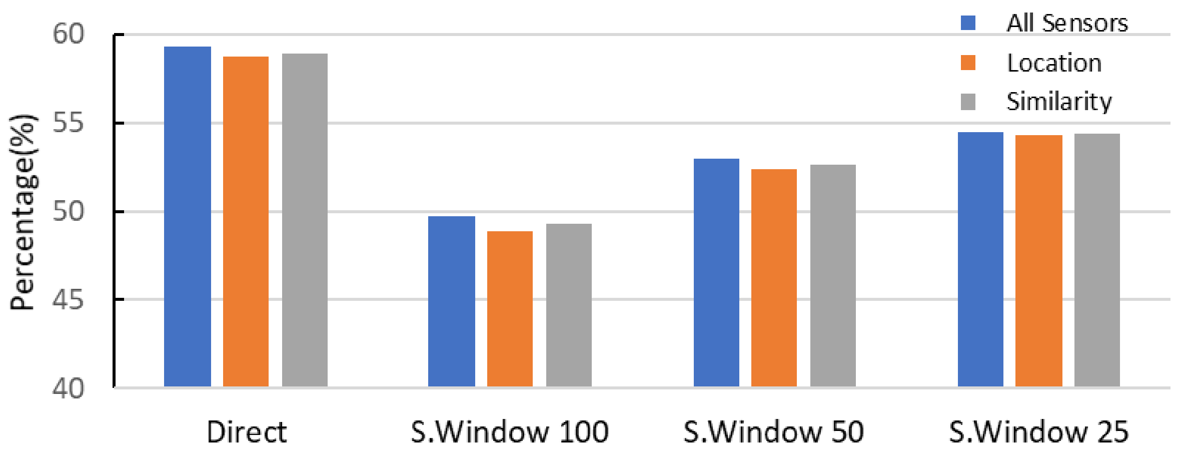

Figure 6.15: Network traffic reduction.

## 6.4 Discussion

Experiment results demonstrate that the presented approach is capable of reducing data sent to the cloud up to 80% without significant loss in accuracy. The sliding window technique increases accuracy even when reduction is carried out on data preprocessed with sliding windows. Window sizes 50 and 100 achieve similar accuracy while window size 25 results in reduced accuracy for all scenarios (Fig. 6.9): all sensors, location-based, and similarity-based. With sufficiently large sliding windows, even a 90% reduction results only in a small loss of accuracy: Fig. 6.6 shows relatively stable accuracy up to a 90% reduction.

However, the sliding window scenarios have a disadvantage of data aggregation on the edge. With the sliding step one, if reduced data is sent to the cloud on every time step, network traffic will still be high as sensor readings will belong to different windows. The sliding step equal to (or larger than) the window size prevents the readings from belonging to different windows and reduces network traffic, but the application scenario must allow for a window length delay.

Classification with reproduced data was slightly better than with reduced data; the difference increased for smaller windows (Fig. 6.9). For larger window sizes 50 and 100, AE and PCA performed similarly, but for window size 25 AE outpreformed PCA for all reduction rates (figures 6.6, 6.7, 6.8). This may be caused by the AE's ability to capture complex relations through non-linear transformations in contrast to PCA's linear transformations which were not as successful.

Location and similarity-based approaches achieved very similar results to all sensors (Fig. 6.5), but the former have the advantage of enabling data reduction on different nodes. Moreover, the location-based approach allows for locating edge nodes closer to the sources of data and thus could be beneficial for geographically distributed scenarios.

The rate of network traffic reduction is lower than the data reduction rate as illustrated in Table 6.4 and Fig. 6.15 due to network overheads. Moreover, network traffic analysis highlighted the need to avoid overlapping sliding windows as they result in high network traffic. However, overlapping windows can still be used for training the networks, AE and classification FFNN, but should be avoided in edge-cloud deployments.

With edge-cloud computing, it is important that the edge nodes are located close to the sources of data. This way, complete raw data from sensors is traveling only from sensors to the edge, and once data is reduced, this reduced data is moving from the edge to the cloud for the final ML task. Consequently, a high-speed connection between the IoT devices and the edge nodes is essential for achieving overall high performance.

The presented approach was evaluated on the HAR task with MHEALTH dataset, but it can be applied for other IoT tasks and data sets. High correlation among different sensor readings will result in high data reduction rates. Moreover, different applications and ML tasks will have different reduction rates depending on how valuable are various parts of the data for that specific application/task.

# Chapter 7

# Conclusion and Future Work

This chapter first provides a high level overview of the presented edge-cloud system for IoT data analytics and examines the main findings in Conclusion section. Next, Future Work section discusses possible extensions of this work.

## 7.1 Conclusion

Proliferation of sensors, wearables, mobile, and other IoT devices has resulted in massive quantities of data, and this trend is expected to continue. Typically, this data is transferred to the cloud or another centralized system possibly thousands of miles away for storage and processing. Then, the results of the analysis are sent from the cloud to the end user for display, process control, or another activity. The value of the collected data relies on the ability to extract value from data, and machine learning has immense potential for value extraction because of its ability to learn from data and provide data-driven insights. This cloud-based workflow is well suited for computationally intensive data analytics tasks such as those involving machine learning because of the computation capacity (storage, memory, CPU, GPU) and scalability the cloud offers. However, transmitting large data quantities among geographically distant locations results in increased network traffic and latencies. With the explosion of connected devices, this will lead to increased latencies and it will put a strain on communication networks.

Consequently, this thesis explores merging edge and cloud computing for machine learning with IoT data with the objective of reducing network traffic and latencies. To reduce the quantity of data sent to the cloud, this thesis uses deep learning, specifically autoencoders. The encoder part of the autoencoder is deployed at the edge to reduce the number of features and data size. Data are then sent to the cloud for further processing. Reduced data can be used directly for the ML task, such as classification, or original data can be restored using the decoder part of the autoencoder in the cloud.

Two computation models are considered: single node and multi-node edge-cloud models. A single-node model uses one edge node for data reduction and sends reduced data to the cloud for final processing. This model was evaluated on human activity recognition from smartphone data. Results show that autoencoders are capable of significantly reducing the quantity of data without considerately impacting ML task accuracy. The accuracy remained almost unchanged when data was reduced by 50% while data reduction of 23% caused only one percent drop in accuracy.

The second approach, the multi-node edge-cloud model, takes advantage of several edge nodes. Three scenarios are considered: all sensors together, location-based, and similarity-based. All sensors together approach actually reduces a multi-mode model to a single-node model as all data is considered together and reduction happens on a single node. Location-based approach groups data according to the IoT device location and data from co-located devices are reduced together on the same edge node. In the similarity-based approach, data is grouped according to the similarity between sensors: for example, all gyroscope data is reduced on the same edge node.

The evaluation of the multi-node edge-cloud model was carried out with data from wearable sensors including accelerometers, gyroscopes, and magnetometers. The HAR task was carried out considering two non-reversible approaches, Autoencoder (AE) and Principal Component Analysis (PCA), and one non-reversible approach, vector magnitude. The results show that data and the corresponding network traffic can be reduced even up to about 80% without significant loss of accuracy if a large sliding window is used in the preprocessing. Location and similarity-based approaches achieve similar accuracy to all sensors approaches, but they can carry out reductions on different edge nodes. The location-based approach has the advantage of transferring data to the edge node closes to the IoT device location what can be beneficial in reducing the network traffic latencies depending on IoT device and edge node location.

Overall, both single and multi-node solutions are capable of significantly reducing network traffic without a major impact on the classification task accuracy. AE achieved slightly better accuracy than PCA, and classification with reproduced data achieved better accuracy than classification with reduced data irrelevant of data reduction algorithm (AE and PCA). Experiments with wearable sensors confirmed the necessity of using the sliding window technique for HAR and showed that small window sizes (25 steps) can result in reduced activity recognition accuracy.

## 7.2 Future Work

The presented single and multi-node edge-cloud computation models demonstrated success on the human activity recognition task and confirmed that network traffic can be reduced by using autoencoders for data reduction on the edge. Moreover, the experiments illustrated the importance of selecting adequate window length for the accuracy of the classification. At the same time, this work opened the door for further investigations with respect to improving the presented edge-cloud models, evaluating them, and exploring new edge-cloud ML directions. Thus, future work will include the following:

- Physical device implementation: All experiments in this theses were carried out through simulations on various workstations and servers with open source data sets. The next step will be to perform the evaluations by assembling a system consisting of real-world IoT devices, edge nodes, and the cloud. It is expected that such a system will impose challenges with respect to communication between systems and may require alternations of the presented approach depending on the computation capability of different components.

- Resource constraints evaluation: Performed simulation experiments did not consider resource limitations. Resource requirements and limitations for edge nodes will be explored to determine the minimum resources requirements for the edge nodes including CPU, memory, and storage.

- Geographically distributed sensors: While this thesis used HAR for the evaluation, the presented edge-cloud system has potential in other applications, specifically those involving geographically distributed sensors such as those present in smart cities. In such an environment, edge nodes will be geographically distributed, and it becomes important to consider both, transfer from edge devices to the edge nodes, and transfer from the edge to the cloud.

- High dimensional data: Experiments with body-mounted sensors considered 7 sensors with 21 readings in total. Future work will consider different ML tasks and different data sources focusing on high dimensional data. Such data may put additional strain on the edge nodes, but the potential benefits are high as it may be possible to achieve high degrees of reduction.

- Data aggregation on the edge: With the sliding window technique, data aggregation happens on the edge node in order to collect a number of readings equal to the length of the sliding window. Future work will explore possibilities to apply a different time-dependent way to assemble this data besides simply concatenating them.

- Edge middleware: Future work will explore developing a compact middleware for the edge nodes to help data pre-processing and reduction as well as communication with the cloud. This middleware would simplify deployment on various edge nodes and enable faster evaluation of different edge-cloud configurations.

The presented edge-cloud architecture for IoT data analytics performed well and showed potential in the human activity recognition use case. However, there is still space for improvement, and evaluations are needed on different data sets and use cases, as has been discussed in this section.

# Bibliography

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT press Cambridge, 2016, vol. 1.

[2] CISCO, "Cisco global cloud index: Forecast and methodology, 2016–2021," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html, 2018, [online], [accessed December 27, 2019].

[3] A. L'heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, "Machine learning with Big Data: Challenges and approaches," *IEEE Access*, vol. 5, no. 5, pp. 777–797, 2017.

[4] H. Cai, B. Xu, L. Jiang, and A. V. Vasilakos, "Iot-based big data storage systems in cloud computing: perspectives and challenges," *IEEE Internet of Things Journal*, vol. 4, no. 1, pp. 75–87, 2016.

[5] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the internet of things," *IEEE Communications Magazine*, vol. 54, no. 12, pp. 22–29, 2016.

[6] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Communications Magazine*, vol. 55, no. 3, pp. 38–43, 2017.

[7] H. El-Sayed, S. Sankar, M. Prasad, D. Puthal, A. Gupta, M. Mohanty, and C. T. Lin, "Edge of things: The big picture on the integration of edge, IoT and the cloud in a distributed computing environment," *IEEE Access*, vol. 6, pp. 1706–1717, 2018.

[8] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[9] R. Roman, J. Lopez, and M. Mambo, "Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges," *Future Generation Computer Systems*, vol. 78, pp. 680–698, 2018.

[10] A. Kumari, S. Tanwar, S. Tyagi, N. Kumar, R. M. Parizi, and K. R. Choo, "Fog data analytics: A taxonomy and process model," *Journal of Network and Computer Applications*, vol. 128, pp. 90–104, 2019.

[11] B. Tang, Z. Chen, G. Hefferman, S. Pei, T. Wei, H. He, and Q. Yang, "Incorporating intelligence in fog computing for big data analysis in smart cities," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 5, pp. 2140–2150, 2017.

[12] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. of the conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[13] H. F. Nweke, Y. W. Teh, M. A. A.-G., and U. R. Alo, "Deep learning algorithms for human activity recognition using mobile and wearable sensor networks: State of the art and research challenges," *Expert Systems with Applications*, vol. 105, pp. 233–261, 2018.

[14] C. Banos, O.and Villalonga, R. Garcia, A. Saez, M. Damas, J. A. Holgado-Terriza, S. Lee, H. Pomares, and I. Rojas, "Design, implementation and validation of a novel open framework for agile development of mobile health applications," *Biomedical engineering online*, vol. 14, no. 2, p. S6, 2015.

[15] F. Mattern and C. Floerkemeier, "From the internet of computers to the internet of things," in *From active data management to event-based systems and more*. Springer, 2010, pp. 242–259.

[16] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future internet: the internet of things architecture, possible applications and key challenges," in *Proc. of the 10th international conference on frontiers of information technology*. IEEE, 2012, pp. 257–260.

[17] F. Wortmann and K. Flüchter, "Internet of things," *Business & Information Systems Engineering*, vol. 57, no. 3, pp. 221–224, 2015.

[18] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[19] IEEE, "IEEE Innovation at Work," https://innovationatwork.ieee.org/real-life-edge-computing-use-cases/, 2019, [online], [accessed February 07, 2020].

[20] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," *In Proceedings of 10th International Conference on Intelligent Systems and Control*, 2016.

[21] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, 2018.

[22] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[23] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[24] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[26] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proc. of the International Conference of Machine Learning*, 2012.

[27] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. of the 25th international conference on Machine learning*, 2008, pp. 1096–1103.

[28] N. Kambhatla and T. K. Leen, "Dimension reduction by local principal component analysis," *Neural computation*, vol. 9, no. 7, pp. 1493–1516, 1997.

[29] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[30] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[31] G. G. Jia, G. G. Han, A. Li, and J. Du, "Ssl: Smart street lamp based on fog computing for smarter cities," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4995–5004, 2018.

[32] P. Hu, H. Ning, T. Qiu, Y. Zhang, and X. Luo, "Fog computing based face identification and resolution scheme in internet of things," *IEEE transactions on industrial informatics*, vol. 13, no. 4, pp. 1910–1920, 2016.

[33] L. Li, K. Ota, and M. Dong, "Deep learning for smart industry: Efficient manufacture inspection system with fog computing," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4665–4673, 2018.

[34] Y. Wang, K. Wang, H. Huang, T. Miyazaki, and S. Guo, "Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 976–986, 2018.

[35] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[36] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[37] A. Sinaeepourfard, J. Garcia, X. Masip-Bruin, and E. Marin-Tordera, "Data preservation through fog-to-cloud (f2c) data management in smart cities," in *Proc. of the 2nd International Conference on Fog and Edge Computing*, 2018.

[38] Y. Jararweh, A. Doulat, O. AlQudah, E. Ahmed, M. Al-Ayyoub, and E. Benkhelifa, "The future of mobile cloud computing: integrating cloudlets and mobile edge computing," in *Proc. of the 23rd International Conference on Telecommunications*, 2016.

[39] N. Mohamed, J. Al-Jaroodi, I. Jawhar, S. Lazarova-Molnar, and S. Mahmoud, "Smartcityware: a service-oriented middleware for cloud and fog enabled smart city services," *IEEE Access*, vol. 5, pp. 17 576–17 588, 2017.

[40] X. Wang, Z. Ning, and L. Wang, "Offloading in internet of vehicles: A fog-enabled real-time traffic management system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4568–4578, 2018.

[41] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multi-tier fog computing with large-scale IoT data analytics for smart cities," *IEEE Internet Things Journal*, vol. 5, no. 5, pp. 677–686, 2017.

[42] S. A. Hossain, M. A. Rahman, and M. A. Hossain, "Edge computing framework for enabling situation awareness in iot based smart city," *Journal of Parallel and Distributed Computing*, vol. 122, pp. 226–237, 2018.

[43] A. M. Rahmani, T. N. Gia, B. Negash, A. Anzanpour, I. Azimi, M. Jiang, and P. Liljeberg, "Exploiting smart e-health gateways at the edge of healthcare internet-of-things: a fog computing approach," *Future Generation Computer Systems*, vol. 78, pp. 641–658, 2018.

[44] P. Ritrovato, F. Xhafa, and A. Giordano, "Edge and cluster computing as enabling infrastructure for internet of medical things," in *Proc. of the 32nd International Conference on Advanced Information Networking and Applications*. IEEE, 2018, pp. 717–723.

[45] M. Chen, W. Li, Y. Hao, Y. Qian, and I. Humar, "Edge cognitive computing based smart healthcare system," *Future Generation Computer Systems*, vol. 86, pp. 403–411, 2018.

[46] H. Wang, J. Gong, Y. Zhuang, H. Shen, and J. Lach, "Healthedge: Task scheduling for edge computing with health emergency and human behavior consideration in smart homes," in *Proc of the International Conference on Big Data*. IEEE, 2017, pp. 1213–1222.

[47] B. Chen, J. Wan, A. Celesti, D. Li, H. Abbas, and Q. Zhang, "Edge computing in iot-based manufacturing," *IEEE Communications Magazine*, vol. 56, no. 9, pp. 103–109, 2018.

[48] X. Li, J. Wan, H. N. Dai, M. Imran, M. Xia, and A. Celesti, "A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 7, pp. 4225–4234, 2019.

[49] D. Wu, S. Liu, L. Zhang, J. T., R. X. Gao, T. Kurfess, and J. A. Guzzo, "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing," *Journal of Manufacturing Systems*, vol. 43, pp. 25–34, 2017.

[50] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019.

[51] E. Zdravevski, P. Lameski, V. Trajkovik, A. Kulakov, I. Chorbev, R. Goleva, N. Pombo, and N. Garcia, "Improving activity recognition accuracy in ambient-assisted living systems by automated feature engineering," *IEEE Access*, vol. 5, pp. 5262–5280, 2017.

[52] M. Z. Uddin, "A wearable sensor-based activity prediction system to facilitate edge computing in smart healthcare system," *Journal of Parallel and Distributed Computing*, vol. 123, pp. 46–53, 2019.

[53] P. Bellavista, S. Chessa, L. Foschini, L. Gioia, and M. Girolami, "Human-enabled edge computing: Exploiting the crowd as a dynamic extension of mobile edge computing," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 145–155, 2018.

[54] S. Y. Nikouei, Y. Chen, S. Song, R. Xu, B. Y. Choi, and T. R. Faughnan, "Real-time human detection as an edge service enabled by a lightweight cnn," in *Proc. of the International Conference on Edge Computing*. IEEE, 2018, pp. 125–129.

[55] C. Y. S. Kin and B. Coker, "Video compression using recurrent convolutional neural networks," *arXiv preprint arXiv*, 2016.

[56] M. N. Fekri, A. M. Ghosh, and K. Grolinger, "Generating energy data for machine learning with recurrent generative adversarial networks," *Energies*, vol. 13, no. 1, p. 130, 2020.

[57] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. Reyes-Ortiz, "A public domain dataset for human activity recognition using smartphones." in *Proc. of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013.

[58] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz, "Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine," in *International workshop on ambient assisted living*. Springer, 2012, pp. 216–223.

[59] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[60] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[61] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

# Appendix A

# Tables

Table A.1: Number of features before and after reduction for reversible approaches (window size 50).

| **Scenario** | **Edge Location** | **Original Features** | **66% Reduction** | **70% Reduction** | **80% Reduction** | **90% Reduction** | **95% Reduction** |
|---|---|---|---|---|---|---|---|
| **All Sensors** | | | | | | | |
| Direct | - | 21 | 7 | - | - | - | - |
| S.Window | - | $21 \times 50$ | 357 | 315 | 210 | 105 | 52 |
| **Location Based** | | | | | | | |
| Direct | L1 | 3 | 1 | - | - | - | - |
| | L2 | 9 | 3 | - | - | - | - |
| | L3 | 9 | 3 | - | - | - | - |
| S.Window | L1 | 3×50 | 49 | 45 | 30 | 15 | 7 |
| | L2 | 9×50 | 149 | 135 | 90 | 45 | 22 |
| | L3 | 9×50 | 149 | 135 | 90 | 45 | 22 |
| **Similarity Based** | | | | | | | |
| Direct | S1 | 9 | 3 | - | - | - | - |
| | S2 | 6 | 2 | - | - | - | - |
| | S3 | 6 | 2 | - | - | - | - |
| S.Window | S1 | 9×50 | 149 | 135 | 90 | 45 | 22 |
| | S2 | 6×50 | 99 | 90 | 45 | 30 | 15 |
| | S3 | 6×50 | 99 | 90 | 45 | 30 | 15 |

Table A.2: Number of features before and after reduction for reversible approaches (window size 25).

| **Scenario** | **Edge Location** | **Original Features** | **66% Reduction** | **70% Reduction** | **80% Reduction** | **90% Reduction** | **95% Reduction** |
|---|---|---|---|---|---|---|---|
| **All Sensors** | | | | | | | |
| Direct | - | 21 | 7 | - | - | - | - |
| S.Window | - | $21 \times 25$ | 178 | 157 | 105 | 52 | 26 |
| **Location Based** | | | | | | | |
| Direct | L1 | 3 | 1 | - | - | - | - |
| | L2 | 9 | 3 | - | - | - | - |
| | L3 | 9 | 3 | - | - | - | - |
| S.Window | L1 | $3\times25$ | 25 | 22 | 15 | 7 | 3 |
| | L2 | $9\times25$ | 76 | 67 | 45 | 22 | 11 |
| | L3 | $9\times25$ | 76 | 67 | 45 | 22 | 11 |
| **Similarity Based** | | | | | | | |
| Direct | S1 | 9 | 3 | - | - | - | - |
| | S2 | 6 | 2 | - | - | - | - |
| | S3 | 6 | 2 | - | - | - | - |
| S.Window | S1 | $9\times25$ | 76 | 67 | 45 | 22 | 11 |
| | S2 | $6\times25$ | 51 | 45 | 30 | 15 | 7 |
| | S3 | $6\times 25$ | 51 | 45 | 30 | 15 | 7 |

Table A.3: Classification accuracy for reduced and reproduced data: 70% data reduction.

| Category | AE | | | | | | PCA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reduced | | | Reproduced | | | Reduced | | | Reproduced | | |
| | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall |
| **All Sensors** | | | | | | | | | | | | |
| S.Window 100 | 99.9% | 1 | 1 | 99.92% | 1 | 1 | 99.89% | 1 | 1 | 99.9% | 1 | 1 |
| S.Window 50 | 99.83% | 1 | 1 | 99.9% | 1 | 1 | 99.82% | 1 | 1 | 99.9% | 1 | 1 |
| S.Window 25 | 99.62% | 1 | 1 | 99.89% | 1 | 1 | 99.55% | 1 | 1 | 99.8% | 1 | 1 |
| **Location Based** | | | | | | | | | | | | |
| S.Window 100 | 99.87% | 1 | 1 | 99.92% | 1 | 1 | 99.86% | 1 | 1 | 99.9% | 1 | 1 |
| S.Window 50 | 99.83% | 1 | 1 | 99.85% | 1 | 1 | 99.81% | 1 | 1 | 99.82% | 1 | 1 |
| S.Window 25 | 99.55% | 1 | 1 | 99.85% | 1 | 1 | 99.37% | 0.99 | 0.99 | 99.8% | 1 | 1 |
| **Similarities based** | | | | | | | | | | | | |
| S.Window 100 | 99.90% | 1 | 1 | 99.92% | 1 | 1 | 99.82% | 1 | 1 | 99.9% | 1 | 1 |
| S.Window 50 | 99.82% | 1 | 1 | 99.89% | 1 | 1 | 99.81% | 1 | 1 | 99.83% | 1 | 1 |
| S.Window 25 | 99.56% | 1 | 1 | 99.88% | 1 | 1 | 99.33% | 0.99 | 0.99 | 99.83% | 1 | 1 |

Table A.4: Classification accuracy for reduced and reproduced data: 80% data reduction.

| Category | AE | | | | | | PCA | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Reduced | | | Reproduced | | | Reduced | | | Reproduced | | |
| | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall |
| **All Sensors** | | | | | | | | | | | | |
| S.Window 100 | 99.88% | 1 | 1 | 99.91% | 1 | 1 | 99.85% | 1 | 1 | 99.89% | 1 | 1 |
| S.Window 50 | 99.8% | 1 | 1 | 99.85% | 1 | 1 | 99.79% | 1 | 1 | 99.88% | 1 | 1 |
| S.Window 25 | 99.6% | 1 | 1 | 99.85% | 1 | 1 | 99.51% | 1 | 1 | 99.79% | 1 | 1 |
| **Location Based** | | | | | | | | | | | | |
| S.Window 100 | 99.87% | 1 | 1 | 99.92% | 1 | 1 | 99.82% | 1 | 1 | 99.87% | 1 | 1 |
| S.Window 50 | 99.77% | 1 | 1 | 99.84% | 1 | 1 | 99.76% | 1 | 1 | 99.78% | 1 | 1 |
| S.Window 25 | 99.49% | 0.99 | 0.99 | 99.82% | 1 | 1 | 99.35% | 0.99 | 0.99 | 99.76% | 1 | 1 |
| **Similarities based** | | | | | | | | | | | | |
| S.Window 100 | 99.89% | 1 | 1 | 99.91% | 1 | 1 | 99.79% | 1 | 1 | 99.81% | 1 | 1 |
| S.Window 50 | 99.77% | 1 | 1 | 99.83% | 1 | 1 | 99.78% | 1 | 1 | 99.79% | 1 | 1 |
| S.Window 25 | 99.52% | 1 | 1 | 99.84% | 1 | 1 | 99.3% | 0.99 | 0.99 | 99.77% | 1 | 1 |

Table A.5: Classification accuracy for reduced and reproduced data: 90% data reduction.

| **Category** | **AE** | | | | | | **PCA** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Reduced** | | | **Reproduced** | | | **Reduced** | | | **Reproduced** | | |
| | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall |
| **All Sensors** | | | | | | | | | | | | |
| S.Window 100 | 99.88% | 1 | 1 | 99.9% | 1 | 1 | 99.8% | 1 | 1 | 99.86% | 1 | 1 |
| S.Window 50 | 99.76% | 1 | 1 | 99.81% | 1 | 1 | 99.76% | 1 | 1 | 99.8% | 1 | 1 |
| S.Window 25 | 99.55% | 1 | 1 | 99.81% | 1 | 1 | 99.48% | 0.99 | 0.99 | 99.75% | 1 | 1 |
| **Location Based** | | | | | | | | | | | | |
| S.Window 100 | 99.87% | 1 | 1 | 99.9% | 1 | 1 | 99.75% | 1 | 1 | 99.8% | 1 | 1 |
| S.Window 50 | 99.72% | 1 | 1 | 99.81% | 1 | 1 | 99.75% | 1 | 1 | 99.75% | 1 | 1 |
| S.Window 25 | 99.42% | 0.99 | 0.99 | 99.78% | 1 | 1 | 99.3% | 0.99 | 0.99 | 99.73% | 1 | 1 |
| **Similarities based** | | | | | | | | | | | | |
| S.Window 100 | 99.88% | 1 | 1 | 99.9% | 1 | 1 | 99.74% | 1 | 1 | 99.79% | 1 | 1 |
| S.Window 50 | 99.74% | 1 | 1 | 99.81% | 1 | 1 | 99.75% | 0.99 | 0.99 | 99.78% | 1 | 1 |
| S.Window 25 | 99.43% | 0.99 | 0.99 | 99.79% | 1 | 1 | 99.25% | 0.99 | 0.99 | 99.74% | 1 | 1 |

Table A.6: Classification accuracy for reduced and reproduced data: 95% data reduction.

| **Category** | **AE** | | | | | | **PCA** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Reduced** | | | **Reproduce** | | | **Reduced** | | | **Reproduce** | | |
| | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall | Accuracy | Precision | Recall |
| **All Sensors** | | | | | | | | | | | | |
| S.Window 100 | 99.79% | 1 | 1 | 99.79% | 1 | 1 | 99.75% | 1 | 1 | 99.79% | 1 | 1 |
| S.Window 50 | 99.74% | 1 | 1 | 99.8% | 1 | 1 | 99.71% | 1 | 1 | 99.76% | 1 | 1 |
| S.Window 25 | 99.48% | 0.99 | 0.99 | 99.76% | 1 | 1 | 99.4% | 0.99 | 0.99 | 99.74% | 1 | 1 |
| **Location Based** | | | | | | | | | | | | |
| S.Window 100 | 99.76% | 1 | 1 | 99.8% | 1 | 1 | 99.71% | 1 | 1 | 99.77% | 1 | 1 |
| S.Window 50 | 99.72% | 1 | 1 | 99.77% | 1 | 1 | 99.68% | 1 | 1 | 99.74% | 1 | 1 |
| S.Window 25 | 99.3% | 0.99 | 0.99 | 99.72% | 1 | 1 | 99.26% | 0.99 | 0.99 | 99.71% | 1 | 1 |
| **Similarities based** | | | | | | | | | | | | |
| S.Window 100 | 99.77% | 1 | 1 | 99.79% | 1 | 1 | 99.71% | 1 | 1 | 99.74% | 1 | 1 |
| S.Window 50 | 99.71% | 1 | 1 | 99.76% | 1 | 1 | 99.68% | 1 | 1 | 99.71% | 1 | 1 |
| S.Window 25 | 99.37% | 0.99 | 0.99 | 99.77% | 1 | 1 | 99.19% | 0.99 | 0.99 | 99.69% | 1 | 1 |

# Curriculum Vitae

| | |
|---|---|
| **Name:** | Ananda Mohon Ghosh |
| **Post-Secondary Education and Degrees:** | Khulna University<br>Khulna, Bangladesh<br>2011 - 2015 B.Sc., Computer Science and Engineering |
| | Western University<br>London, ON, Canada<br>2018 - 2020 M.E.Sc., Electrical and Computer Engineering<br>Vector Institute: Collaborative Specialization in Artificial Intelligence |
| **Related Work Experience:** | Teaching Assistant and Research Assistant<br>The University of Western Ontario<br>2018 - 2020 |

**Publications:**

- Ghosh, A.M. and Grolinger, K., Edge-Cloud Computing for IoT Data Analytics: Embedding Intelligence in the Edge with Deep Learning. IEEE Transactions on Industrial Informatics, 2020 [Revise and resubmit].

- Fekri, M.N., Ghosh, A.M. and Grolinger, K., Generating Energy Data for Machine Learning with Recurrent Generative Adversarial Networks. Energies, 13(1), 2020.

- Ghosh, A.M. and Grolinger, K., 2019, May. Deep Learning: Edge-Cloud Data Analytics for IoT, In Proc. of the IEEE Canadian Conference of Electrical and Computer Engineering, pp. 1-7, 2019.