Deep Learning: Edge-Cloud Data Analytics for IoT

Ananda M. Ghosh, Katarina Grolinger
Department of Electrical and Computer Engineering
Western University
London, ON, Canada N6A 5B9
{aghosh45, kgroling}@uwo.ca

***Abstract*—Sensors, wearables, mobile and other Internet of Thing (IoT) devices are becoming increasingly integrated in all aspects of our lives. They are capable of collecting massive quantities of data that are typically transmitted to the cloud for processing. However, this results in increased network traffic and latencies. Edge computing has a potential to remedy these challenges by moving computation physically closer to the network edge where data are generated. However, edge computing does not have sufficient resources for complex data analytics tasks. Consequently, this paper investigates merging cloud and edge computing for IoT data analytics and presents a deep learning-based approach for data reduction on the edge with the machine learning on the cloud. The encoder part of the autoencoder is located on the edge to reduce data dimensions. Reduced data are sent to the cloud where there are used directly for machine learning or expanded to original features using the decoder part of the autoencoder. The proposed approach has been evaluated on the human activity recognition tasks. Results show that 50% data reduction did not have a significant impact on the classification accuracy and 77% reduction only caused 1% change.**

***Index Terms*—Edge Computing, Deep Learning, IoT, Data Analytics, Autoencoder**

I. Introduction

In recent years, the explosion of sensors, wearables, mobile and other Internet of Things (IoT) devices has been changing how we live and work. Today, there are over six billion connected devices, and Gartner estimates that this number will grow to over 20 billion by 2020 [1]. According to the McKinsey Global Institute, IoT could generate $11 trillion dollars a year in economic value by year 2025 [2]. Connected devices have created the foundation for smart systems such as smart watches, buildings, cities, cars, appliances, and smart grids. Each day, people are becoming more dependent on smart and connected devices and, although only a small portion of IoT data is presently used [2], new connected devices are continuously emerging.

Commonly, IoT devices are equipped with sensors that detect and measure changes in their environment. Recorded data are exchanged over the network for storage, processing, or control. To realize the benefits of intelligent/smart IoT systems and extract value from collected data, data analytics is essential. Traditional approaches to data analytics transfer these data to the cloud for storage and processing, and subsequently deliver results and/or data to software applications.

For example, data from a smart home are transferred to a data centre, possibly thousands of miles away, for storage and processing, and then back for display on devices within the home. Attempting to transfer all IoT data to the cloud for processing will put a strain on communication networks and increase latencies caused by the data transfer over large distances. On the other hand, those connected devices are limited with respect to the analytic that they can perform because of factors such as limited computation power, storage capacity, memory and battery life.

Edge computing (EC) [3]–[5] has been proposed as a way to address these challenges by physically pushing the computation away from the centralized system and to the edges of the network and sources of data. Performing part of the computation on the device itself or on a node close to the source of data can reduce data that need to be sent to the cloud and consequently reduce latencies and improve response time. Edge Servers [3], servers that reside on the edge of a network, often act as a connection between a private network and the Internet. They can be used for data and computation offloading as well as for multimedia content provision. Moreover, edge servers 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. Tang *et al.* [6] suggested that edge computing could be used to extract features and reduce the number of featured sent to the cloud.

Deep Learning (DL) has recently shown good results in a variety of domains, especially when large data quantities are available. It has great potential in the IoT context because it can carry out representation learning by transforming data into hierarchical abstract representations that enable learning good features. Deep autoencoders, a type of deep neural networks, can be used to learn efficient data encoding in an unsupervised manner.

This study explores the use of deep learning, specifically autoencoders, for data reduction in the edge-cloud IoT data analytics context. Autoencoders are suitable for this task because once an autoencoder is trained, the encoder part of the network can be employed on the edge to reduce data that are sent to the cloud. The decoder part of the trained network can then be deployed on the edge to perform dimensionality reduction. Since training deep networks, including autoencoders, is computationally expensive, it still needs to be done on the cloud.

In this work, different autoencoder architectures are compared with respect to how much they reduce data and how they affect data analytics tasks. Two main scenarios are considered: in the first scenario, autoencoders are used for data reduction and the cloud machine learning (ML) task is carried out with the reduced data. In the second scenario, autoencoders on the edge reduce data similar to the first scenario, but on the cloud, the reduced data are first expanded (original data restored) before performing the ML task. The experiments are carried out on the task of human activity recognition with the data from smart-phones. In both scenarios, autoencoders have demonstrated the ability to reduce network traffic, without significantly impacting machine learning task accuracy.

The remainder of the paper is organized as follows: Section II provides background concepts and Section III reviews related work. The methodology is described in Section IV, and the results are presented in Section V. Finally, Section VI concludes the paper.

II. Background

This section introduces deep learning and dimensionality reduction.

A. 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 [7]. In recent years, DL has gained popularity because it demonstrated an ability to learn complex models and perform representation learning [8]. This type of learning uses data representations rather than explicit data to perform learning: data are transformed into hierarchical abstract representations that enable learning of good features, which are consequently used for the ML tasks.

Figure 1 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.

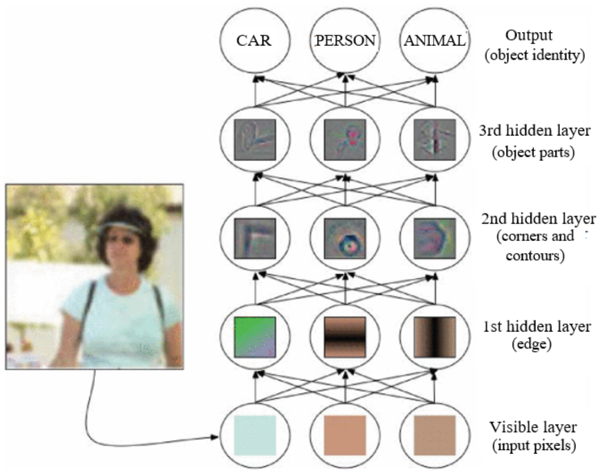
CAR
PERSON
ANIMAL
Output (object identity)
3rd hidden layer (object parts)
2nd hidden layer (corners and contours)
1st hidden layer (edge)
Visible layer (input pixels)

Fig. 1: Deep Learning [9]

Deep learning 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. The proposed approach uses unsupervised learning, specifically autoencoders, to reduce data sent to the cloud.

Autoencoders (AE) [10] are neural networks trained to reproduce input vectors as output vectors. The input vector $x_1, x_2, ..., x_n$, is compressed through the hidden layers which have a fewer number of neurons than the input layer, into lower dimensional subspace and then uncompressed to reproduce the input as output. The objective is to learn a representation or data encoding for a data set.

In experiments, in addition to using autoencoders for data reduction, we use DL, particularly feedforward neural networks for human activity recognition.

B. Dimensionality Reduction

Machine learning problems commonly encounter a large number of features, or attributes, which is common in a variety of machine learning tasks, especially with large data sets. However, as the dimensionality increases, it becomes more difficult and time consuming to work with the data set and the effectiveness of the ML algorithm decreases [8]. Therefore, *dimensionality reduction* is used to reduce the number of features or attributes under consideration.

Principal Component Analysis (PCA) [11] is a widely used linear dimensionality reduction technique. It uses orthogonal transformations to convert a set of possibly correlated features 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 constraint that it is orthogonal to the preceding components. Dimensionality reduction is achieved by using only the first *n* principal components.

Autoencoders can also be used for feature reduction; they perform non-linear transformations whereas PCA carries out linear transformations. Moreover, autoencoders ability to extract higher level features makes them suitable for dimensionality reduction as demonstrated by Wang *et al.* [12]. The number of neurons in the hidden layers controls the degree of dimensionality reduction, whereas the number of the hidden layers impacts how encoded features are computed. The experiments presented in our work compared accuracy achieved with autoencoders and PCA.

III. Related Work

In recent years, edge computing has been attracting significant attention from both research and industry due to its promise to reduce latencies and network traffic, improve user experience, and reduce reliance on cloud [3], [5], [13], [14]. Edge computing shares many attributes with fog computing which refers to processing data close to data sources. Therefore, this section considers both, edge and fog.

Several studies have discussed edge computing concepts, potentials, advantages, and disadvantages [6], [15] without demonstrating edge performance with experiments. For example, El-Sayed *et al.* [5] presented the big picture on the integration of edge, IoT, and cloud in a distributed computing environment. They discussed edge computing functional capabilities and identified challenges such as device selection, automated task allocation, computation offloading, communication overhead, mobility management, and security and privacy. Additionally, they compare edge computing with cloud and multi-could computing focusing on highlighting edge advantages. Finally, El-Sayed *at al.* conclude that edge computing technologies can provide distributed data processing for IoT applications overcoming drawbacks of traditional centralized processing.

The second group of related works presents experiments that demonstrate edge computing capabilities. Sinaeepourfard *et al.* [14] proposed fog to cloud (F2C) data management architecture incorporating the data preservation block with the objective of providing faster data access than the cloud. To illustrate 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.* [16] 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.

Li *et al.* [17] 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 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 fog nodes and higher-level layers on the server. Performed experiments demonstrate high defect detection accuracy, decreased load on the central servers, and reduced overall computation time; however, latencies and communication overhead were not analyzed.

Jia *et al.* [18] proposed a Smart Street Lamp (SSL) system based on fog computing for smart cities with the objective of reducing maintenance periods, decreasing 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 improving 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.

Study by He *et al.* [19] presented a multi-tier fog computing model with large-scale data analytics services for smart city applications. The multi-tier fog consists of ad-hock 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 setup, ad-hock nodes are lower resource devices such as Raspberry PIs and desktops, whereas higher resource servers are used as dedicated nodes. Presented experiments compare different architectures in respect to service utility and blocking probability.

A fog-enabled real-time traffic management system investigated by Wang *et al.* [20] offloads computation to the fog nodes with the aim of minimizing 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.

The discussed studies either examine edge and fog possibilities, advantages and challenges, [6], [15] or are concerned with a very specific use case scenario without specifically addressing data analytics or machine learning. In contrast, our work explores employing machine learning approach for data reduction on the edge nodes to decrease network traffic and latencies caused by data transfer to cloud for machine learning.

IV. Methodology

This section first introduces the edge-cloud architecture for data analytics. Next, data reduction with autoencoders and edge-cloud computation models are discussed.

A. Edge-cloud Architecture for Data Analytics

The proposed edge-cloud architecture for data analytics is depicted in Figure 2. 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 edge and possibly connect to cloud for integration purposes [20].

Sensors are capable of high-frequency sampling: for example, voltage and frequency sensors can record thousands of readings per second. Because of this, quantity of data that need to be transferred is very large. Attempting to transfer 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 proposed edge-cloud architecture for data analytics presented in Figure 2 is to reduce quantity of data transferred to the cloud in a way suitable for machine learning. While other edge computing

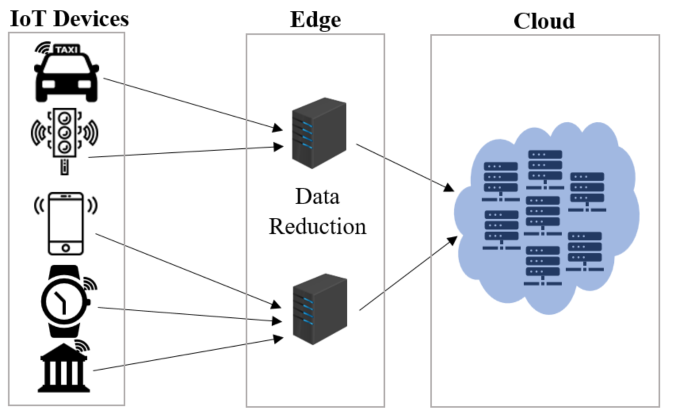

Fig. 2: Edge-cloud Computing Architecture for Data Analytics

solutions also reduce data transfer, the solution proposed 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 [8]. This paper 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 2, 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 fog 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.

B. Data Reduction with Autoencoders

For data reduction on edge nodes, this study uses autoencoders. As already mentioned, autoencoders are capable of dimensionality reduction by learning hierarchical data representations. As illustrated in Figure 3, autoencoders take input data and process them through a number of hidden layers. A 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 network is located on the edge, and the decoder part is on the cloud. This way, when high-dimensional data arrive to 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 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 cloud or GPU-enabled devices.

C. Edge-cloud Computation Models

This work considers three computation models as illustrated in Figure 4. 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 edge nodes 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, recall, and F1-score. 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 performed and the reduced data are sent to the cloud. Instead of carrying out an 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 we are concerned with the accuracy of the reconstructed signal. If the autoencoders inner layers have too few neurons, in other words, an attempt was made to compress data too much, accuracy of the reconstructed signal will be

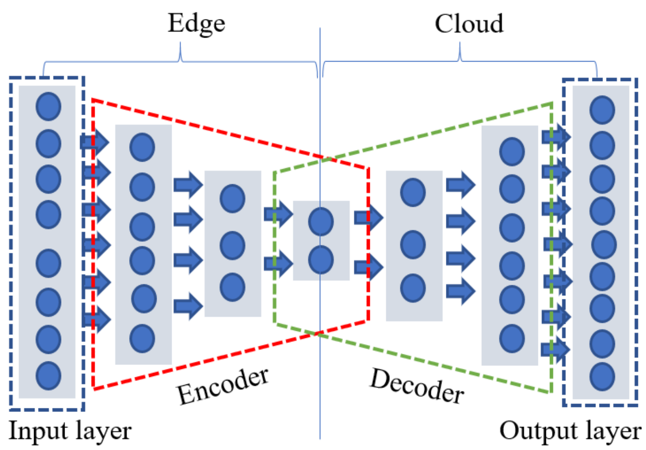

Fig. 3: Autoencoder for Dimensionality Reduction in Edge-cloud Architecture

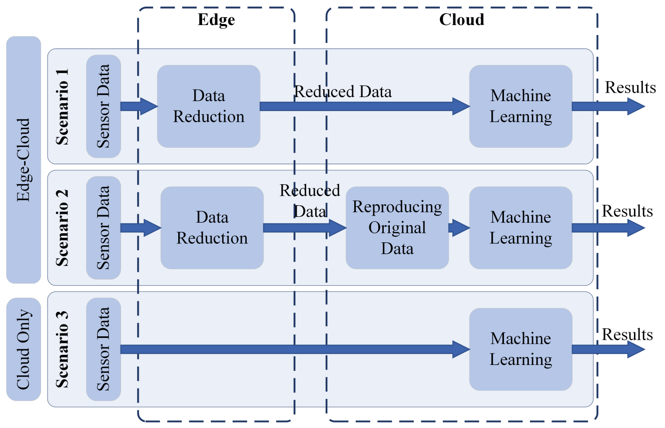

Fig. 4: Edge-cloud Computation Models

affected. Examples of error measures for reconstruction accuracy include Mean Squared Error (MSE) and Mean Absolute Error (MAE). If the reconstructed data are of low quality, the accuracy of the final ML task will be reduced.

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 proposed 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 feature selection, 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 in this paper evaluated the impact of autoencoder architecture on ML task accuracy and data reduction.

V. Evaluation

This section first introduces the machine learning task used for the evaluation and the data set, then describes the experiments and discusses their results.

A. Machine Learning Task and Data Set

The proposed 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 [21].

This study uses publicly available human activity data set [21], [22]. Anguita *et al.* created this data set from accelerometer and gyroscope readings at a sampling rate of 50Hz [21]. Raw data were preprocessed, and the time signals were sampled in a fixed-length sliding window, resulting in the final data set with 561 features describing human activities. 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 30% for testing. All values were normalized by scaling to the range [0,1]. Note that in the proposed approach, the training is carried out on a single centralized system, and then the encoder part is deployed onto an edge node and the decoder part on the cloud.

B. Experiments

Three types of experiments were performed corresponding to the three scenarios from Figure 4:

- 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 the comparison purposes.

Autoencoder'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.

All three scenarios use a neural network for the classification: the number of input neurons corresponds to the number of dimensions after reduction and the number of output neurons is six (six classes). Hidden layers were added to improve classification accuracy. Other network parameters include:

- Optimizer: Adam
- Epocs: 1000
- Loss: MSE (Mean Squared Error)
- Activation: ReLu + Softmax

To evaluate the classification, *accuracy*, *Mean Squared Error* (MSE) and *Mean Absolute Error* (MAE) were used. Accuracy represents the ratio of correctly classified samples to the total number of samples, whereas MSE and MAE are calculated as follows:

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

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

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

C. Results and Discussion

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 warranted 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 first (Scenario 3 from Section IV-C); results are presented in Table I. 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 V-B: Figure 5 shows the results. Values on the horizontal axis 256, 128, and 64 show number of features after data reduction. Scenario 2 is indicated as *encoded* and Scenario 3 as *decoded*. In the figure, *'original'* refers to the classification with original data (Scenario 1) 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, 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 is 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. Higher accuracy with encoded data than with decoded data can be explained by the effects of the curse of dimensionality [8] and inaccuracies introduced by the decoding process.

Figure 6 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 it is the case with

TABLE I Classification Accuracy

Accuracy	MSE	MAE
0.9545	0.01311	0.01752

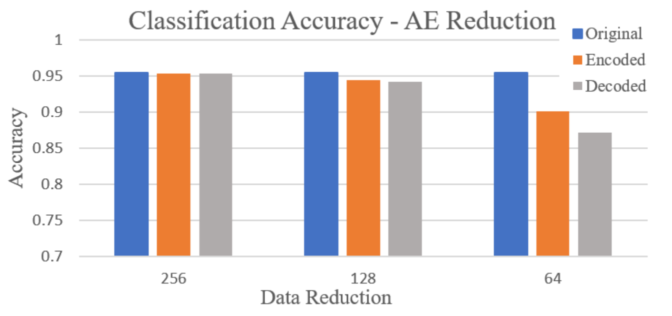

Fig. 5: Classification accuracy with autoencoder data reduction

autoencoders, 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 accuracy of ML algorithms [8].

Reduction using an autoencoder and PCA is compared in Figure 7; an autoencoder (AE) with encoded 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 more significant drop in accuracy with both, PCA and AE. Nevertheless, even with 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 8 shows the size of original data compared to reduction to 256, 128 and 64 features. It can be seen that data reduce from 11.2MB for original data to 5.754MB for 256 features, 2.877MB for 128, and 1.4386MB 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 we were able to reduce the number of features

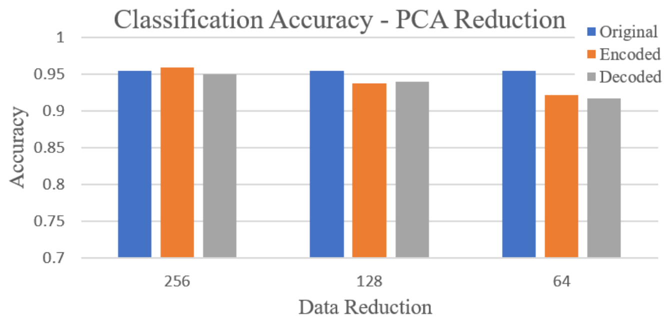

Fig. 6: Classification accuracy with PCA data reduction

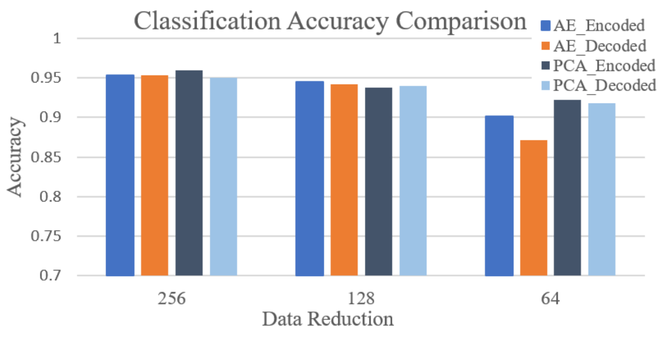

Fig. 7: Autoencoder and PCA comparison

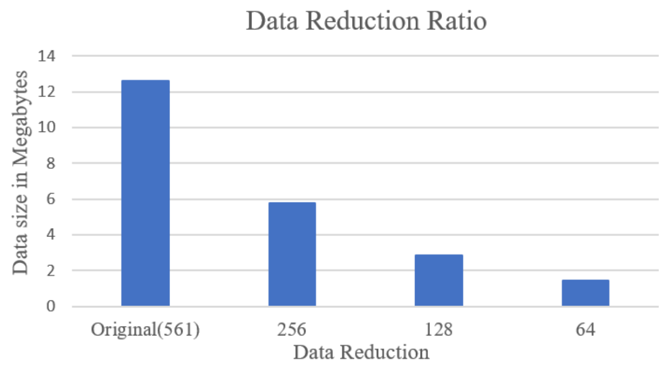

Fig. 8: Data size for different feature reduction

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

VI. Conclusion and Future Work

Proliferation of sensors, wearables, mobile, and other IoT devices has resulted in massive quantities of data, and this trend is expected to continue. Attempting to transfer all these data to the cloud for storage and processing will result in increased network traffic and latencies.

To address these challenges, this paper proposes to combine edge and cloud for IoT data analytics. To reduce quantity of data sent to the cloud, the study uses deep learning, specifically autoencoders. The encoder part of the autoencoder is deployed on 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. The proposed approach was evaluated on human activity recognition from smartphone data. Results show that autoencoders are capable of significantly reducing the quantity of data without significantly impacting ML task accuracy.

Future work will explore the application of the proposed approach for different ML tasks and with different data sets focusing on high dimensional data. Moreover, kernel PCA and incorporating time-dependencies typical of IoT data will be explored.

References

[1] P. Middleton, J. F. Hines, B. Tratz-Ryan, E. Goodness, D. Freeman, M. Yamaji, A. McIntyre, A. Gupta, D. Rueb, and T. Tsai. Forecast: Internet of things endpoints and associated services. *Gartner*, 2016.

[2] M. James, M. Chui, P. Bisson, J. Woetzel, R. Dobbs, J. Bughin, and D. Aharon. The internet of things: Mapping the value beyond the hype. *McKinsey Global Institute*, 2015.

[3] X. Sun and N. Ansari. EdgeIoT: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29, 2016.

[4] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck. Mobile edge computing potential in making cities smarter. *IEEE Communications Magazine*, 55(3):38–43, 2017.

[5] 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*, 6:1706–1717, 2018.

[6] 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*, 13(5):2140–2150, 2017.

[7] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[8] A. L'heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz. Machine learning with Big Data: Challenges and approaches. *IEEE Access*, 5(5):777–797, 2017.

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

[10] P. Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proc. of International Conference of Machine Learning*, 2012.

[11] N. Kambhatla and T. K. Leen. Dimension reduction by local principal component analysis. *Neural computation*, 9(7):1493–1516, 1997.

[12] Y. Wang, H. Yao, and S. Zhao. Auto-encoder based dimensionality reduction. *Neurocomputing*, 184:232–242, 2016.

[13] M. T. Beck, M. Werner, S. Feld, and S. Schimper. Mobile edge computing: A taxonomy. In *Proc. of the 6th International Conference on Advances in Future Internet*, 2014.

[14] 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 IEEE 2nd International Conference on Fog and Edge Computing*, 2018.

[15] Z. Hao, E. Novak, S. Yi, and Q. Li. Challenges and software architecture for fog computing. *IEEE Internet Computing*, 21(2):44–53, 2017.

[16] 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 23rd International Conference on Telecommunications*, 2016.

[17] L. Li, K. Ota, and M. Dong. Deep learning for smart industry: Efficient manufacture inspection system with fog computing. *IEEE Transactions on Industrial Informatics*, 14(10):4665–4673, 2018.

[18] 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*, 14(11):4995–5004, 2018.

[19] 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*, 5(5):677–686, 2017.

[20] 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*, 14(10):4568–4578, 2018.

[21] 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 European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2013.

[22] D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proc. of International Workshop on Ambient Assisted Living*, 2012.