2022: A Computational Odyssey

Towards a Deeper Understanding of Clustering Streaming Human Activity Recognition Data

by

Martin Woo

A thesis submitted to the

School of Computing

in conformity with the requirements for

the degree of Master of Science

Queen's University

Kingston, Ontario, Canada

August 2022

Abstract

Global integration of network-connected devices such as smartphones and smartwatches into commonplace life have inadvertently created various fields researching the plethora of data streams now made readily available. Human Activity Recognition (HAR) is one such example and understanding the hierarchy of human movements can have positive implications in elderly care, physiotherapeutic therapy, surveillance, and general healthcare, among others. However, the typical modalities for deep analysis - pure supervised learning - are limited to the classes provided during training and are incapable of relaying any hidden knowledge residing outside of the trained labels. Unsupervised learning methods, such as clustering, are not only well equipped to handle high-density data streams and address the challenges inherently prevalent in this medium of data, but are also capable of automatically extracting varying levels of abstraction from the data represented by the relationships within and between clusters.

This thesis presents a journey that begins with the hypothesis that time-series IoT data can be clustered to effectively identify human activity. However, experimental results ultimately evolves the question into *what else is needed to allow effective clustering of HAR data for identifying human activity.* A subsequent exploration tries to answer two additional research questions: can deep learning approaches add a

higher level of insight for time-dependent human movement sequences and uncover unintuitive knowledge regarding the actions, and can visualization allow for better understanding on the nature of human activities? The field of clustering with machine learning methods for HAR time-series IoT (Internet-of-Things) data collected from sensors is an undeveloped region of study, and the work present in this thesis contributes valuable content to the area of research. The journey begins with an initial experiment on applying stand-alone clustering algorithms onto a HAR data stream, advances to the development of hybrid algorithms to extract features and recognize human activity from IoT sensor data, and concludes with an analysis of visualization algorithms that can be used for improving the interpretation of the clustering methods. Autoencoder implementations are justifiably introduced after the realization the standalone clustering models are incapable of fully uncovering the HAR patterns, but their contribution in extracted feature quality are luckluster, leading to the decision to employ temporal-based machine learning methods to include temporal components during data analysis. The use of hybrid temporal-based architectures finally achieves optimal results, extracting and returning exceptional temporal features for form distinct clusters. Our journey ends with the visualization of the temporally-reduced clusters and the conclusion on the necessity for supervised-learning-based temporal feature extraction to supplement unsupervised learning algorithms for clustering highly complex temporal-based HAR data.

Acknowledgments

I offer my most sincere gratitude to my supervisor, Dr. Farhana Zulkernine, for her constant guidance and invaluable advice throughout this long and highly stressful project. Her unwavering perseverance to read my rambling lines of text into the (very) late hours of the night has not gone unnoticed and is appreciated more than words can describe. The knowledge she bestowed upon me is invaluable, and without her counseling I would most certainly be a lesser academic.

I would like to thank the world's most fluffiest and actively aggressive cat I have ever had the pleasure of being bitten by: Momo. The physical damage her single tooth causes is negligible but it is the persistence that makes it endearing. Her constant drive to dominate over all life forms inspired me and gave me the strength to finish this thesis.

Finally, my most heartfelt appreciation to my friends and family who supported me through the highest of ups and the lowest of downs, and in several instances, battles of dominance with a single-toothed cat.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The rapid advancements in technological research and development have inadvertently forced the ubiquity of network connected systems of Internet of Things (IoT). The integration of web accessibility in these systems enables them to communicate with other systems and the cloud for data ingestion and analysis [61], but comes at the cost of requiring mechanisms capable of addressing the inherent challenges present in the volatile data streams. Methodical navigation of the following problems will allow for effective knowledge extraction critical to the advancement of machine cognition [45, 93]:

1. Functionally infinite size: Data streams are practically infinite while flowing at high speeds. Continuous accumulation of large data can therefore lead to data swamps with little to no information extracted from it, while consequently consuming large storage space.

2. Dynamic trends: The dynamic nature of data streams requires methods capable of adapting to changing trends. At the very least, the models must be capable of learning new information without sacrificing old information.

3. Unique data structures: Different types of multi-modal and hybrid data require corresponding processing and analytic pipelines.

4. Outliers: Live data is expected to contain outliers. The model or analytic pipeline should be able to detect and address these outlying data points.

Considering the above challenges, stream data processing systems must incorporate computational models that can quickly analyze large volumes of data arriving at high rates, identify and extract information necessary to enable advance machine cognition, learn from the data, and return the results in real-time for further processing of the data if necessary. Models should be robust enough to capture the underlying data patterns from noisy streaming data and identify outliers as observed in a real-life use case scenario, but be sensitive enough to recognize the larger concepts while still learning the small deviations in the patterns.

The domain of Human Activity Recognition (HAR) is one field of research that must consider these practical implementation challenges including the difficulty of effectively collecting raw data for conducting analysis [110, 43]. Expanding on the knowledge for this specialty enables improvements of various application domains such as healthcare [61, 99, 137, 124], surveillance [113, 122, 75], remote care for the elderly [61, 107, 65, 60], and understanding the human body through monitoring and deeper analysis [7]. For example, one of the most urgent motivations for precise activity monitoring is fall detection. Falls are one of the predominant health risks affecting the elderly and is the second leading cause of death with an estimated 640,000 fatal falls occurring each year [84]. Reportedly, 30% of people above the age of 65 have fallen each year and this number increases to 40% for those above 70 [125]. The fear of falls causing debilitating injuries is a strong motivator for long-term care facility

admissions [117] - however the shortage of staff demand for automated patient monitoring to improve quality of life and enable prompt notification of abnormal patient activity to arrange urgent assistance. Deeper understanding on the intricacies of human activities - in particular, the transitions and underlying relationships between actions - can improve the detection of falls and subsequently lower the risk of prolonged injuries arising from delayed emergency response. A more recent application for HAR-based analysis is in physiotherapy, where the primary goal is rehabilitation of physical injuries through maintained adherence of prescribed exercises. Constant in-person monitoring is unfeasible, and a variety of influence such as the home environment, cognitive status, or unintentional deviations from the prescribed treatment scheme can cause delayed progression in rehabilitation [118]. Automated activity recognition and intricate knowledge regarding the specificities of the appendages in question enables effective and consistent treatment through positive feedback when the patient performs the exercises exactly as prescribed. Intricate understanding on the nature of human movements and producing tangible research results by translating the concepts into working applications provides market value for businesses with investments in the sectors.

1.1 Problem Description

For HAR, a large amount of accurate measurements need to be recorded over a period of time. The stand-alone sensor devices that were used prior to multi-sensor devices (such as smartphones or smartwatches) were cumbersome and restricted the subject from natural movement. To prevent accidental collisions with the sensing devices, the subject made conscious adjustments to their movements, resulting in

unnatural sequences of action and ultimately hindering accurate data transcriptions of the intended activity [110, 43]. The invention of smartphones and smart wearable devices with built-in sensors allowed for an exponential increase in data gathering convenience and data accuracy [16, 108], which in turn triggered a re-interest in HAR research. The ability to generate, collect, and analyze data at high efficiencies has led to the compilation of numerous HAR datasets [110]. Human movements possess a natural flow from activity to activity, making it difficult for models to discern the boundary between the beginning of an action and the end of another action. To provide an intuitive example, take the sequence of movements starting from standing up to sitting down. From a high level perspective, we can simply classify the entire sequence as 'sitting down', but if we elaborate on the details, we can discern several smaller sub-movements: 1) Standing - from the initial segment of the sequence, 2) Stand-sit transition or Sitting - the movement from standing to sitting, and 3) Sit - the final seated position. Typically, sub-movement 2 (the transition) would not be considered a distinct action by sensor systems, because transitional movements are too quick and numerous to properly classify. Therefore, HAR datasets do not contain labels for these small transition points, which leads to the question whether we can detect such transition points. For example, where within the IoT data sequence does sub-movement 1 (standing) end and sub-movement 3 (sit) begin? Are our HAR models capable of learning the knowledge representing movement transitions and correctly analyze the sequences? Since time-series data can be structurally manipulated to fit specific model architectures, can we change the way the model receives the sequences to improve performance?

Another problem residing in HAR sequence data concerns the possibility for two

sequences to be considered the same activity but with noticeably different sensor values. This manifests when there is a change in subject position (e.g. the subject is still walking but now the elevation is higher or lower then before because they ascended or descended a flight of stairs), when the sensors pick up the natural randomness within human movements (from our inability to move completely meticulously), or from the imperfect precision of the sensors. The analytical models within the ML pipeline must be capable of recognizing these scenarios and accurately identify the sequences as the same action and generate either a singular encompassing cluster or two spatially adjacent clusters to represent the similarities between the sequence features. Consequently, experimentation on all aspects of the machine earning pipeline must be explored to best determine the ideal combination of methods to address the general stream processing and HAR-specific problems in addition to maximize analytical performance.

1.2 Challenges in IoT-based HAR

Much of the existing research (on sensor IoT data from smartphones and smartwatches equipped with Inertial Measurement Units (IMU) such as accelerometer and gyroscope) focuses on supervised learning methods [3]. Unsupervised learning approaches such as clustering are rarely seen in the literature concerning HAR machine learning applications. Understandably, if the purpose of the research is identifying the actions, then supervised methods are ideal. However, the models can only classify data corresponding to the given label - they are incapable of automatically extrapolating subtle micro-actions within the overarching patterns like unsupervised methods

can. Take the previous example about the 'sitting down' sequence: classification algorithms can only differentiate between the two typically labelled actions, standing and sitting, while observations of the results from a clustering algorithm can provide insights regarding the transitional movements depending on the relationships of the clusters. In other words, unsupervised clustering has the potential to dive deeper in the data and extract knowledge that is intuitively observed by humans but not recognized by machines.

HAR datasets are complex as they generally contain time-stamped sequential data from multiple IoT components which can be high-dimensional, high volume, and noisy. Various machine learning (ML) methods exist for dimensionality reduction and temporal and spatial feature extraction, and are occasionally paired with visualization methods to present the learned knowledge for deeper analysis and intuitive understanding. However, a thorough comparative study providing model information regarding 1) the capacity for deeper understanding of the learned patterns on HAR data, 2) model efficacy, 3) ability to handle streaming IoT data, and 4) capability to identify transitions of activities does not exist.

1.3 Research Objectives

Since supervised machine learning algorithms are limited in their capability to explore the given data beyond the provided labels, we aim to analyze human activity using unsupervised clustering methods. Then, from the resulting clusters we can manually ascertain knowledge regarding the nature of human movements and establish a hierarchy of actions by exploring the relationships between and within activity clusters. Our initial hypothesis questions whether we can effectively cluster IoT data to identify

human activities, however preliminary results sparked a reframing of the hypothesis and launched an exploration to answer two additional research questions. This thesis present a pathway of systemic exploration of many ML models for the ultimate objective of clustering HAR data to identify human activities and extract transitional knowledge from sequences of timestamped streaming IoT data. We progress from simple statistical clustering algorithms to deep neural network architectures with the intention to develop a novel ML pipeline that improves clustering quality. Supervised learning strategies are introduced into the ML pipeline to enable deep feature processing to aid the unsupervised methods with analysis. For knowledge representation, we explore a number of visualization algorithms for high-dimensional data clustering to assess their ability to present human activities and their transitions.

Our evaluation on the effectiveness of each ML method largely depends on the scores returned by the performance metrics corresponding to either method of learning process. For any model in the analytical pipeline that employs unsupervised clustering, we evaluate using Normalized Mutual Information Score (NMI) and Adjusted Rand Index (ARI), two standard cluster quality metrics seen throughout literature. Architectures that embody supervised learning use accuracy instead. Regardless of the approach, if the score is high, then the model is considered to have performed well. The value thresholds differentiating between good and poor performance are referenced from literature that contains either similar datasets or model architectures. For the purpose of this thesis, we aim to maximize the metric scores across all models.

The following research questions are addressed in this thesis:

1. Can time-series IoT data be clustered to effectively identify human activity using unlabelled data?

   (a) What other components are required to improve clustering quality?

2. Can the transitions between activities be learned? What ML methods provide useful insights?

3. Can visualization methods provide a better understanding of the data?

1.4 Contributions

Our thesis contributions are as follows:

1. A literature review is provided on the state of stream clustering algorithms.

2. Multiple experiments are conducted to evaluate a variety of clustering models namely WCDS, FISHDBC, and K-means, on a benchmark dataset, MobiAct v2.0, to determine their efficacy in handling simulated streaming information. Performance of the models is evaluated based on clustering quality (ARI and NMI), time for processing, and deployability to low-powered devices.

   (a) Spatial feature extraction models - namely Convolutional Neural Networks (CNNs) - are evaluated and the features were then used in clustering,.

   (b) Temporal Feature Extraction (TE) models are used to learn and extract temporal features and apply the reduced representation to clustering. We experiment with 4 evolutions of the TE design and evaluate the ideal iteration on the basis of performance, computational cost, computational time, and level of dataset preparation for data processing.

   (c) We explore various Autoencoders (AE) for automatic feature extraction and dimensionality and noise reduction.

(d) Goodness of the extracted reduced features from AE models are first evaluated using classification models and then used in clustering. A comparison with the TE models shows that in general, the features returned by the supervised learning methods outperform those outputted by AE algorithm. Hybrid architectures combining the two methods leverages the power of both designs and performs optimally in regard to classification and clustering.

3. We provide an analysis of the HAR datasets using state-of-the-art visualization methods and compare their effectiveness in explaining intricacies. We implement, demonstrate and compare three modern dimensionality reduction and subsequent visualization algorithms.

Our journey begins with a comprehensive literature review on the current state of the stream clustering field and the evolution of algorithms which led up to the current iteration. The subsequent work on the independent literature clustering models leads to our conclusion that more powerful machine learning algorithms (i.e. artificial neural networks) are necessary to untangle the complexities and intricacies present in HAR data. From here, our baseline for clustering performance is established: clustering without prior processing gives around 0.5 ARI and NMI. An array of AE implementation are tested to determine if the data will benefit from feature compression. A 2D CNN + AE, LSTM AE, DEC model, and a hybrid Conv AE + LSTM architecture are evaluated. Experimental results demonstrate that the inclusion of AEs can sometimes be detrimental to performance, but the combination of an AE with a supervised learning method can achieve very high scores (0.93 classification accuracy and around 0.76 NMI/ARI). Additional experiments show the effects of highly

volatile transitional actions and how they negatively influence cluster quality despite taking up only 3% of the total dataset. These tests show the necessity for temporal-based processing, and the following set of experiments focus on extracting temporal features. A base hybrid architecture consisting of 1D CNN and LSTM layers is constructed and various improvements on the model is made to evaluate the limits of its processing capabilities. Depending on the method of data preparation, this temporal extraction model is able to achieve 93-98% classification accuracy and 0.67-0.99 ARI and NMI. Further experimentation explore the effects of window length on feature extraction quality, with results showing that while longer windows seem to perform better, expert knowledge of the dataset is needed to prevent unintentional adverse effects. The journey ends with an overview of several visualization algorithms as an example of how visualizing the clusters can provide better insights into the data. A flowchart of our gradual exploration is shown in Figure 1.1.

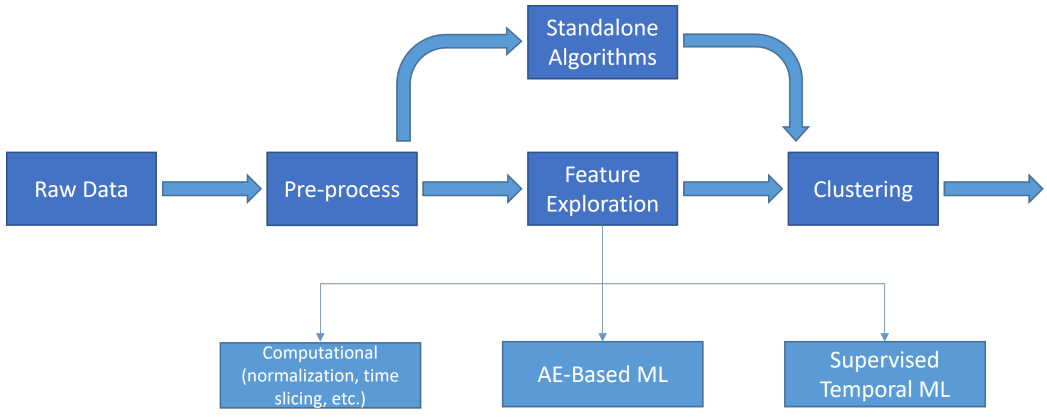

Figure 1.1: Flow diagram of our exploration process

1.5 Organization of Thesis

In Chapter 2, we provide necessary background concepts and literature review on stream clustering algorithms, our ML architectures, and AE algorithms. We discuss statistical models and evaluate their performances in Chapter 3. The model exploration is presented in Chapter 4 and 5 where the Autoencoder (AE) architectures are illustrated in Chapter 4 and the Temporal Extraction (TE) models are demonstrated in Chapter 5. Several visualization algorithms used to create topological representations of the data are described in Chapter 6. Finally, Chapter 7 concludes the thesis and outlines the future work.

Chapter 2

Background

In this Chapter, we explain the fundamentals of stream clustering. A summary of the field is given by Guha et al [49] and several other recent works by Ghesmoune [46] and Silva et al [111]. The paper presented by Mansalis et al [88] additionally provides experimental analysis of some of the algorithms. Chapter 3 presents a comprehensive literature review of the field and evolution of stream clustering algorithms. The relevant background knowledge for our implementations are presented in the corresponding background section of the chapter.

This chapter is presented as follows: Section 2.1 highlights the core concepts shared between all stream clustering algorithms and Section 2.2 presents background knowledge on dynamic learning, Convolutional Neural Networks, Autoencoders, and Long Short-Term Memory. Section 2.3 describes the evolution of stream clustering algorithms. Literature reviews on Autoencoder, Long Short-Term memory, and dynamic learning models are given in Section 2.4. The chapter concludes with a summary discussed in Section 2.5.

2.1 Stream Clustering Fundamentals

The fundamental concepts needed for all implementations of stream clustering algorithms include window definition to create partitions of the data stream [17, 46], preprocessing the data in the partitions [106, 111], and extraction and feeding the preprocessed data into machine learning models [106, 46]. Data streams suffer from noisy and missing data. Some of the common preprocessing techniques applied to data streams involve filtering [106], summarizing data within time windows [139, 106], computing average values for numeric data [88, 106], and modeling data distribution [49, 106]. For later processing of the data using machine learning algorithms, a general approach is to create a summary of the data within time windows and use the summary in data modeling algorithms instead of the raw data [46, 49]. In this section, we describe the common concepts about data stream preprocessing for clustering at a later stage.

2.1.1 Partitioning Data Streams

The three most popular methods in the literature to create partitions of streaming data using window definitions are as follows [111].

- Sliding window: This is the simplest window implementation, where a window of fixed size k, captures elements between the interval of the current time t, and t - k + 1. From this definition, each iteration reads in a fixed number of observations, with the previous data discarded in favour of the current ones [17]. This window model is ideal for problems where only the most recent data is desired.

- Damped window: This window model assigns a weight depending on when the observation first arrived, with the most recent data associated with the highest weight which eventually decreases based on an exponential function. In this model, past observations are not discarded completely [111].

- Landmark window: The landmark window is defined by a starting point (the landmark) to the current time. Multiple landmarks can be established depending on the context of the problem. If our landmark is t=1, this means we take into account all the points in the stream [46].

2.1.2 Processing stages

In the context of stream clustering, the following processing stages are typically applied.

- Online: This is the intuitive method for stream clustering, where the algorithm processes the stream in a single-pass [46]. The model must be capable of maintaining the clusters it creates as new entries arrive.

- Online-offline: This is a method first proposed by Aggarwal et al [1] which consists of two phases - the online data summarization and the offline clustering. The online phase extracts the core information within that observation and the offline phase uses that information to create the final clusters.

2.1.3 Data Summarization

The importance of data summarization can be realized by reviewing the necessity for the algorithm to process the stream quickly and accurately. Given the infinite nature

of data streams, clustering models cannot save large amounts of data in memory. Instead, an abstraction must be used through data summarization. Some of the most common summarization methods are listed below.

- Clustering Feature (CF): This groundwork summary was first presented by Zhang et al in BIRCH [135]. If given N d-dimensional data points, the CF is defined as:

$$CF = (N, \vec{LS}, \vec{SS})$$

  where $\vec{LS}$ is the linear sum of the observations and $\vec{SS}$ is the sum squared of the observations.

- Micro-clusters: the micro-cluster is based on the clustering feature CF with the notable difference being the inclusion of temporal information. The micro-cluster is defined as follows:

$$MC = (N\vec{LS}, \vec{SS}, S\vec{T}^t, S\vec{S}^t)$$

  where $S\vec{T}^t$ is the sum of the timestamps and $S\vec{S}^t$ is the sum-squared of the timestamps.

- Core-micro-clusters: This summary was first presented by Cao et al [20] in their DenStream algorithm, where they combined density-based clustering with

stream data clustering. A core-micro-cluster is defined as follows:

$$CMC = (w, c, r)$$

where w is the weight, c is the center of the micro-cluster, and r is the radius.

- Grids: The data are abstracted into grid cells, which depends on several parameters such as cell size and cell density. Essentially, the model is able determine the population of each cell and access its summary.

2.1.4 Clustering Approaches

There are several general approaches to stream clustering as listed below.

- Density-based: Clusters are created by observing areas of high-density separated by areas of low density. Typically, these algorithms are able to find arbitrary-shaped clusters in addition to identifying outliers. As such, the algorithm automatically discovers the number of clusters usually based on the given parameters. The most famous density-based algorithm is DBSCAN [37].

- Partitioning: these algorithms split the observations into n clusters, where n is an input parameter given by the user. The data is partitioned by minimizing error which is defined as the inter-cluster distance (typically sum of squared error). The most popular cluster partitioning algorithm is K-means [83].

- Grid-based: These approaches are another implementation of density-based algorithm, where the densities are represented by grid population [46]. The input space is split up into grids and the data is partitioned based on its density.

- Hierarchical methods: Hierarchical clustering is primarily performed by two approaches. In the top-down approach, all the observations begin in one large cluster which is then partitioned as the data hierarchy moves down [46]. In the bottom-up approach, each observation first belongs in its own cluster, which are then recursively merged as the the model moves up the hierarchy [46]. Depending on the parameters, the number of clusters can be determined by a predefined threshold when the model splits the clusters [88].

- Neural network-based: Neural network models are being explored for stream clustering which pose a number of challenges when dealing with time-series noisy unlabelled IoT data that can change over time. The models need to be trained with similar data [46].

2.2 Background

This section will cover the background information pertaining to the concepts used in this thesis. We begin with the concept of dynamic learning, continue with knowledge about Convolutional Neural Networks, and finish with Autoencoder and Long Short-Term Memory.

2.2.1 Dynamic Learning

One method for addressing the problems associated with stream clustering is to define an algorithm capable of learning and re-learning knowledge as more information is seen [44, 48]. Dynamic learning is a machine learning technique that enables a model to adapt towards the changing trends typically observed in streaming data. Let us explore an example where we task a machine learning algorithm to classify flowers

for our florist business. Since we established our store in the springtime, we initially trained our model to classify spring flowers - we trained it on a large dataset and finally deploy it onto our website. The problem arises when winter comes around and spring flowers are no longer in season. Our model does not perform well unless it is re-trained on winter flowers and re-deployed online. This can be expensive and time consuming. Incremental learning aims to solve this problem by slowly adapting to new concepts while potentially forgetting previously outdated concepts. The term incremental learning is vague and is used interchangeably with other definitions to describe different techniques. Some of the definitions that have been implied or used are listed below:

1. After training a model on the initial dataset, changes to a new dataset are identified by the model as it performs incremental learning. This represents the learning (or unlearning) of features in subsequent data feeds [48].

2. The model adapts to changing trends in the data stream in real-time. This is typically used to identify concept drift, the phenomenon where concepts change overtime represented by a shift in the relationship between input and output [44].

Dynamic learning implementations are not included in the scope of this thesis and therefore the subject is not truly explored. Any dynamic learning models mentioned in this chapter can reserved for investigated at a later time in future work.

2.2.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) scans and partitions the data to summarize data in the partitions over multiple layers to extract increasingly abstract and deeper

set of spatially organized features. These features are then fed into a fully connected neural network for classification or clustering. Alternating convolutional and pooling layers enable transformation and summarization of the data. The whole network is trained to optimize the parameters to reduce the classification error or improve cluster quality. Referring to Fig 2.1, we can define another set of neurons, A, that takes a sub-sequence of 2 points and calculates particular features from them. This types of layer is called a convolutional layer. A model example with stacked convolutional layers and a max-pooling layer is shown in Fig 2.2. The loss function used to train CNNs using supervised learning approach with labeled data is given below.

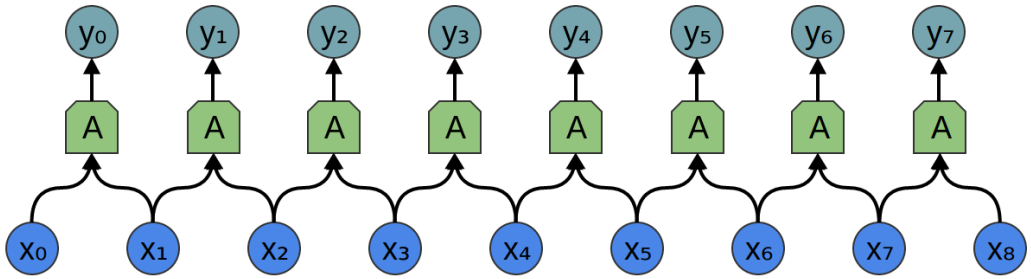

Figure 2.1: CNN with one convolutional layer consisting of a set of neurons, A

The formal definition for the outputs in Fig 2.1 is simply

$$y_n = A(x_n, x_{n+1}, x_{x+2}, ...) \tag{2.1}$$

and the definition for $A(x)$ is

$$A(x) = \sigma(Wx + b) \tag{2.2}$$

where W is the weight matrix for the neurons in the layer and b is the bias.

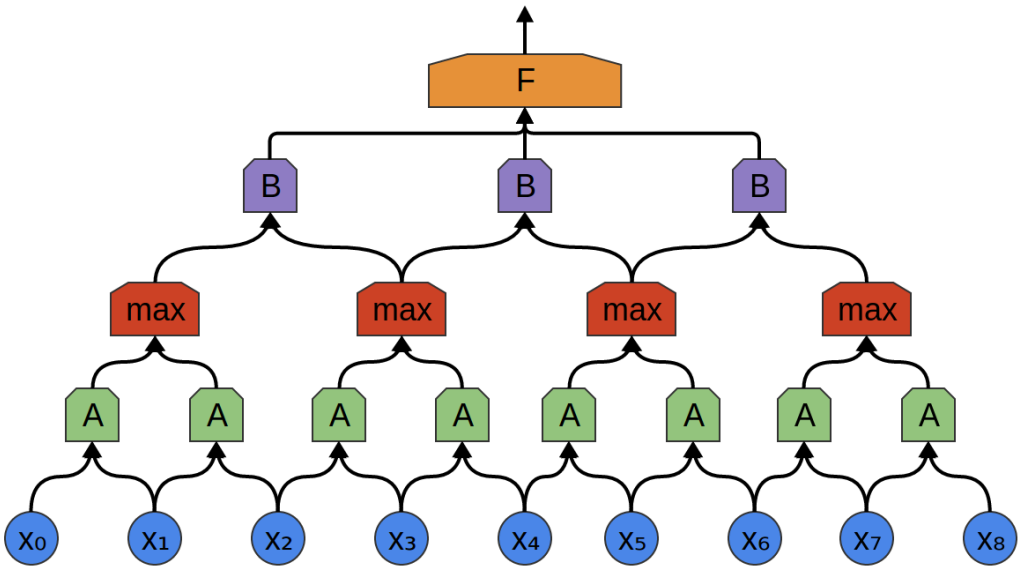

Figure 2.2: Stacked Convolutional layers with Max pooling connected to a fully-connected layer

2.2.3 Autoencoder

An Autoencoder consists of an encoder part and a decoder part. It is trained using unsupervised learning. During training, the encoder extracts and remembers key features as weights from a set of unlabeled training examples of the form $\{x_1, x_2...x_n\}$, where X_i represents the input vector. At the same time, the decoder learns to reconstruct the input in the form $\{\hat{x_1}, \hat{x_2}...\hat{x_n}\}$, where $\hat{X}_i$ represents the reconstructed vector [126] while the complete two part structure tries to minimize the reconstruction loss indicating how close the reconstructed output is to the original. The encoder part helps to reduce data dimensions by learning core representational features that effectively retains the distinctive features of the original data and can reproduce the same with minimum error using the decoder. The ultimate goal of an AE is to extract the key features from the data and reduce data dimensionality.

2.2.4 Long Short-Term Memory

Classic neural networks (think traditional multi-layer perceptions) attempt to imitate processes within human cognition, but they lack the capability to take previous data into account when analysing current inputs. The first implementation that captures this feature is the Recurrent Neural Network (RNN). This network evolved from the classic feed-forward networks to include an internal memory state capable of propagating persisting information forward. Figure 2.3 shows the structure of a typical RNN. On the left is the compressed model of a single unit which unfolds to the the model on the right. The input, 'x', is fed into the network, which generates an output of 'o'. The 'v' loop allows information to be passed through time steps in the network. In a RNN, the structure of a unit is relatively simple, comprising of just a 'tahn' activation function. However, this simplicity suffers from a serious problem. When we train a neural network, we typically use the gradient descent algorithm to find a local minimum of the loss function. To update the weights for each unit, information is back-propagated through the network. For RNNs, the process is more complex because 1) the information has to travel through time, meaning previous timesteps are fed into the next timesteps, and 2) the loss value can be calculated for each timestep. Each node that participates in the loss calculation should have their weights updated, but for RNNs (and their time-series data), this means *every* node from all previous timesteps. The problem appears when node weights (and subsequent gradients) become very low. As small values are multiplied with other small values (weights), the value decreases very quickly. At a certain point, the gradient becomes so small that weight change becomes practically nonexistent for nodes further back in time, resulting in either greatly increased training time, or incomplete

training due to lack of weight updates. This is the vanishing gradient problem [53]. We could alternatively frame the problem as the model not being able to look back far enough to extract information needed the current sequence. For example, if we had the sentence 'I have never eaten Italian food before... I would like to try ___', the logical prediction would be some kind of Italian food, but to realize our choices, we would have to go back down the sequence to find the context. However, because of the potential for vanishing gradients, RNNs could have a hard time if the contextual information is too far back. From this high level perspective, RNNs have trouble handling 'long-term dependencies' as a result of the vanishing gradient problem.

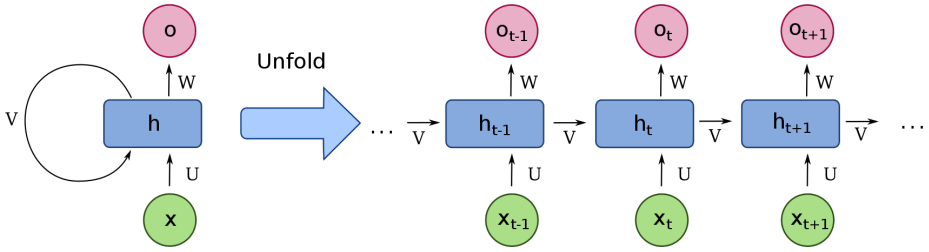

Figure 2.3: Simple RNN unit

LSTM networks were proposed explicitly to counter the vanishing gradient problem [54] and handle long-term dependencies. They achieve this by regulating what information gets passed through using several gates inside each unit. The structure of a typical unit is shown in Fig 2.4. The core of LSTM networks is the cell state, as represented by the horizontal line going from C_{t-1} to C_t. The cell state holds information the LSTM currently has in memory. The gates are methods the model uses to control what gets passed through each iteration of the cell state. The first decision the LSTM makes is how much information to discard from the cell state.

This is achieved with the 'forget gate layer' using a sigmoid function. At time t, the layer takes in h_{t-1} and x_t to output a number between 0 and 1 for each value in the cell state C_{t-1}, with 0 representing 'completely forget' and 1 representing 'completely remember'.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.3}$$

The model then decides what information to store into the cell state with a two-part process. The first phase has the 'input gate layer' decide the values to update with another sigmoid function. The second phase, a tanh layer, returns the optimal values $\tilde{C}_t$ that can be added to the new state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i \tag{2.4}$$

$$\tilde{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_C \tag{2.5}$$

In order to update the old cell state (C_{t-1}) to create the new state (C_t), we multiply C_{t-1} by f_t to forget the previously determined information. Then the model adds the new state values scaled by a factor.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.6}$$

The output of the cell is based on the cell state but filtered through a tanh function

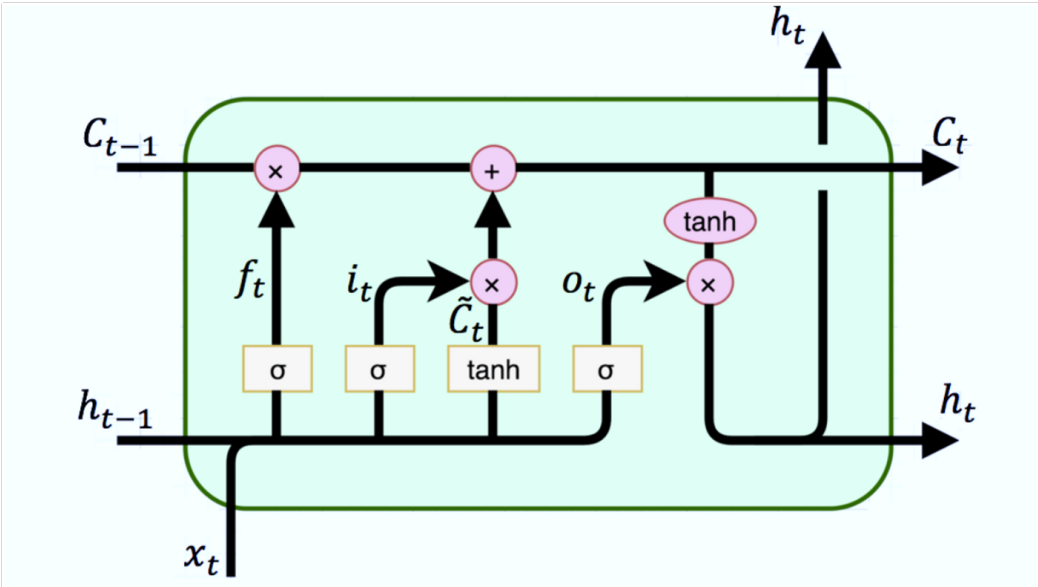

Figure 2.4: Regular LSTM unit

and multiplied by the output of a sigmoid gate to output only the parts that it want.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o \tag{2.7}$$

$$h_t = o_t * tanh(C_t) \tag{2.8}$$

2.3 Evolution of Stream Clustering: A Review

There is a vast literature on data stream-clustering. In this section, we present a literature review to highlight specifically the evolution of research in this area and discuss the two prominent types of models namely the statistical machine learning and neural network-based approaches, which have demonstrated superior performance in clustering unlabeled high speed high volume streaming data. The evolution of algorithms explains the similarities between some of the above algorithms such as sharing the same core implementations, while differing mainly in the data processing or analytical

operations. However, one aspect which almost all the algorithms share, is the metrics used for measuring performance. Most of the existing work in the literature apply unsupervised learning methods for clustering assuming that the data labels are not available for training the models. Cluster validation metrics such as Normalized Mutual Information (NMI) score and Adjusted Rand Index (ARI) score are commonly used as the validation metrics, which we explain later in this paper. Other metrics as presented in Table 2.3 include accuracy in clustering, purity, recognition rate, V-measure, CMM, RAND, SSE, Precision, Recall and F-measure. Table 2.5 provides the equations for these metrics.

A number of frameworks and machine learning models have been implemented in the pursuit of discovering the ideal clustering model for streaming data. Generally, models that are used in stream clustering can be categorized as: artificial neural network (ANN) models and statistical models. We present the state-of-the-art research from both categories that are relevant to stream clustering in the literature review. Tables 2.1 and 2.2 shows a summary of the the related work. A comparative report of the performances of some of the more relevant models is given in Tables 2.3 and 2.4 based on the published data.

2.3.1 Statistical Approaches

Statistical models have traditionally been applied to data clustering problems. The recent statistical machine learning models apply unsupervised learning approaches. The notable common aspect of these models is learning to extract the key data features from the input data and store this representation using an optimal data structure. The learned stored patterns are compared to the input data to determine the

similarity and find groups or clusters in the data.

Foundational Statistical Models

Decades of research on data clustering has resulted in a wealth of influential algorithms and machine learning models that serve as the foundation of many cutting-edge frameworks. One such algorithm is BIRCH [135]. An ongoing problem with high-velocity streaming data is the speed and volume of the data which makes it challenging to process every data point at a high speed. BIRCH addresses this problem by introducing a data structure called Clustering Feature (CF) vectors that stores a summary of the input data. A CF structure represents a triple (N, $\vec{LS}$, $\vec{SS}$) where N is the number of inputs, $\vec{LS}$ is the linear sum of points, and $\vec{SS}$ is the sum squares of the data.

$$CF = (N, \vec{LS}, \vec{SS}) \tag{2.9}$$

In the optional offline version of BIRCH, the popular K-means [83] algorithm is used to refine the clusters. The online-offline implementation can be applied as a recurring process, which has been adopted in majority of the recent stream clustering models.

BIRCH has been extended to propose the CluStream algorithm by adapting the CF data structures to additionally store temporal information for creating an improved summary of the incoming streaming data which is referred to as 'micro-clusters' [1]. The micro-cluster structure adds two additional features to the CF vector: $\vec{ST}^t$, the sum of timestamps, and $\vec{SS}^t$, the sum-squared of the timestamps. The algorithm incorporates an offline phase to perform K-means clustering on the

cluster summaries to create 'macro-clusters'. This model, CluStream, sets the precedence for using time as a vector in stream data analysis as well as concretely cementing the online-offline approach as a valid method in stream clustering.

$$CF = (N, \vec{LS}, \vec{SS}, \vec{ST^t}, \vec{SS^t}) \tag{2.10}$$

DenStream [20] builds upon the CluStream algorithm and adopts many of its design choices. It implements the micro-cluster structure with all three of BIRCH's data features in addition to the online-offline clustering architecture. The core difference between DenStream and its predecessor is the ability to assign decaying weights to data samples. By deleting micro-clusters with weights that fall below a certain threshold, the algorithm removes outdated clusters and is able to maintain an updated representation of the data. With DBSCAN [37] that is used in the offline clustering step, DenStream is able to use cluster density to find arbitrary shaped clusters as opposed to the elliptical shaped ones found by K-means in CluStream.

2.3.2 State-of-the-art Statistical Models

Recent research on statistical clustering approaches presents cutting-edge architectures that are capable of addressing unsolved issues in stream clustering, while simultaneously improving and refining clustering results. ClusTree [69] is built from the CluStream architecture and presents a self-adaptive *anytime* stream clustering algorithm that is able to cluster stream data starting at any point of the steam. It is also parameter free and can adapt to the speed of the stream, while remaining vigilant for outliers and concept drift. This is possible because of the hierarchical balanced tree that ClusTree uses to store the micro-cluster summaries, which guarantees quick

Table 2.1: A summary of the related work discussed from the existing literature.

Algorithm	Dataset	Approach	Underlying Method	Processing
BIRCH Zhang et al. 1997	Pixel Classification Synthetic Clusters	Hierarchical	K-means	Single-pass
GNG Fritzke 1997	Synthetic	Neural Network	Neural Gas	Online
CluStream Aggarwal et al. 2003	KDD-CUP'99 KDD-CUP'98	Partitioning	K-means	Online-Offline
DenStream Cao et al. 2006	Synthetic Clusters KDD-CUP'99 KDD-CUP '98	Density-based	DBSCAN	Online-Offline
SOINN Furao and Hasegawa 2006	Optdigits facial recognition	Neural Network	Self-Organizing Map	Online
ESOINN Furao et al. 2007	Optdigits facial recognition	Neural Network	SOINN	Online
ClusTree Kranen et al. 2011	Forest Covertype Synthetic	Any-time	K-means/DBSCAN	Online-Offline
AING Bouguelia et al. 2013	Optdigits MNIST Bag of words	Neural Network	Growing Neural Gas	Online
LB-SOINN Zhang et al. 2014	Artificial AT&T face	Neural Network	SOINN	Online
HA-Stream (I-HAStream) Hassani et al. 2015 (2016)	Synthetic KDD-CUP'99 ICML'04 Forest Covertype	Density / Hierarchical	Agglomerative	Online
SNCStream (SNCStream+) Barddal et al. 2015 (2016)	KDD-CUP'99 Electricity Forest Covertype	Social Network Model	Network	Online
G-Stream Ghesmoune et al. 2016	KDD-CUP'99	Neural Network	Growing Neural Gas	Online
IDEStream Khan et al. 2016	Electricity consumption	Density-based	DBSCAN	Online

access to all nodes regardless of the level of granularity.

HAStream [51] has been proposed following the success of density-based stream clustering algorithms (such as DenStream) to address the problem of clusters having differing densities. HAStream has the ability to adapt its density threshold according to the batch of incoming data. This approach allows the algorithm to recognize certain clusters that might be missed by the other density-based models for having a density

Table 2.2: Continuation of Table 2.1.

Algorithm	Dataset	Approach	Underlying Method	Processing
WCDS Cardoso et al. 2017	KDD-CUP'99 Forest Covertype	Neural Network	Agglomerative	Online-Offline
GPA Liu et al. 2018	OSUPEL Opportunity CAD14	Bayesian generative model	Network	Offline-Online
FISHDBC Dell'Amico 2019	Blobs Synth Power Consumption Handwritten Letters	Density-based Hierarchical	DBSCAN	Single-pass
DenSOINN Xu et al. 2019	KDD-CUP'99 HAR Forest Covertype digits	Neural Network	SOINN	Online-Offline
HOFCM Yu et al. 2019	MNIST other images	Neural Network	Autoencoder Fuzzy-C-means	Offline-Online
AdapVAE Zhao et al. 2019	MNIST REUTERS-10k STL-10	Neural Network Bayesian Nonparametric	VAE BNP DPMM	Offline-online
DeepStreamCE Chambers et al. 2020	modified CIFAR-10	Neural Network	CNN Autoencoder	Offline-Online
VAE Jakubowski et al. 2021	HRM C-MAPSS	Neural Network	AE VAE	Offline
VAE Krajsic et al. 2021	Loan application dataset	Neural Network Clustering Algorithm	VAE OCSVM IF LOF	Offline-online
GNG-L Wu et al. 2021	4 static datasets 15 Synthetic evolving streams	Neural Network	Growing Neural Gas	Online
FIDC Laohakiat et al. 2021	Synthetic PenDigits Waveform Spambase Landsat Image Segmentation Multi Features	Density-Based	Incremental Density-based	Online-Offline
IMOC-Stream Attaoui et al. 2022	Forest CoverType Sensor Powersupply HyperPlan	Genetic Algorithm	Ant-Tree	Online-Offline

below a pre-set threshold. In the online step, a typical data summarization structure is used such as the micro-clusters. In the offline step, the algorithm creates a cluster hierarchy based on the densities and returns a dendrogram for easy analysis. Shortly after HAStream, I-HAStream [50] was proposed by making it incremental through the maintenance of the inherent Minimum Spanning Tree (MST) and eliminating the need for completely recalculating the structure.

Social Network Clusterer Stream (SNCStream) [12] and subsequently, SNCStream+ [13] have been proposed to cluster streaming data based on the concept of social networks where similar individuals (or groups) aim to seek out other similar individuals (or groups). Through consistent graph vertices rewiring, natural sub-graphs emerge to represent clusters of data. The models adopt the use of micro-clusters and decaying weights as a means to address concept drift and data evolution. Instead of following the usual method of combining an online-offline phase, SNCStream presents a completely online model having the ability to incrementally update the micro-clusters. The incremental update feature is also implemented by I-HAStream [50]. This design allows for the elimination of the offline clustering phase and all the disadvantages that comes alongside to implement either the K-means or DBSCAN algorithms.

IDEStream [63] is also considered a real-time clustering algorithm but it is different with regard to how it calculates density clusters, and how it uses a gamma mixture model as opposed to something like DBSCAN. IDEStream eliminates the offline clustering phase by merging it with the online data summarization phase and applying incremental ensemble clustering to the incoming data within each time window. The clustered results are aggregated with the returned clusters from the previous time window and the process repeats until termination.

Another notable stochastic model is FISHDBC [32]. As mentioned previously, the framework is an approximation of HDBSCAN, which, in turn, is based on DBSCAN. HDBSCAN improves on DBSCAN by allowing the algorithm to cluster hierarchically through the use of a minimum spanning tree (MST). FISHDBC adds on to this by allowing the MST to be updated incrementally, as opposed to being rebuilt each time when new data points are passed in. The incremental updates are enabled through building an *approximation* of the MST rather than completely reconstructing it.

Fuzzy Incremental Density-based Clustering (FIDC) [73] adopts the incremental density-based clustering method, a technique for processing data in a single pass by first creating a local cluster representation during the online phase, then by determining the final clusters in the offline phase. The problem with this method is the potential for inconsistent clusters due to the order of data coming in. FIDC addresses this by employing fuzzy clustering - points are assigned to multiple local clusters. A new algorithm called Modified Valley seeking Algorithm (MVSA) automatically estimate outlier threshold for each density-peak, thus allowing for adaptive and flexible clustering assignment.

A 2022 paper introduced IMOC-Stream [8], a hybrid framework combining the genetic algorithm, Ant-Tree, with a Multi-Objective Clustering method (MOC). Ant-Tree [9] imitates the assembling behaviour of ants, an observed phenomenon where ants attach themselves to an established structure and elongate the connected by subsequently attaching to other ants. A tree is built motivated by this behaviour, where the position of 'ants' (data) and where they connect to within the structure are determined by the similarities between nearby ants. MOC finds clusters by taking

the optimal clustering solution from multiple results and constructing an ideal partition. IMOC-Stream combines these methods and improves on time computation by modifying Ant-Tree to work on streaming data and optimizing when the algorithm applies the genetic functions.

2.3.3 Artificial Neural Network Approaches

Artificial Neural Network models typically apply iterative tuning of model parameters using some machine learning algorithm to train on large datasets. The models implement a network architecture consisting of nodes and links where the weights along the links connecting the nodes represent learned patterns as cluster centroids or prototype vectors. During the training phase, models extract key features from the input data and compute the distance of the feature vector from the weight vectors to adjust the weights to align with the cluster centroid. During the testing phase, the input data pattern is assigned to the cluster that is the nearest.

Foundational ANN Models

Neural network models must implement the appropriate learning algorithms based on the nature of streaming data. High speed incoming streaming data requires a robust and efficient model to quickly process the data. Otherwise, there would be a risk of losing data relevance. Neural networks typically take time to train through multiple iterations. Therefore, a common approach to developing neural network models for processing streaming data is to train the models in offline mode with stored data and later deploy the model to process online data [93, 30]. The other approach allows dynamic learning in online mode but then applies an adjustment phase to discriminate

and adjust the overall cluster shapes afterwards [93, 30].

Competitive neural networks are typically used for the second method of implementation. Much of the modern networks in this branch are built of established neural stream clustering models, such as Kohonen Self Organizing Maps (SOM) [66], Growing Neural Gas (GNG) [39], and Self Organizing Incremental Neural Networks (SOINN) [40]. The fundamental concept behind these algorithms and the reason why they are called 'competitive networks' is the competition that is sparked between the output nodes whenever a new input pattern is presented to the input nodes. The neuron that is structurally closest to the input becomes the winner and each time the winning node is gradually moved closer to the input data point eventually leading to positioning the centroids at the centers of each cluster for the given dataset. The trained network with the centroids converged at the cluster centers, becomes capable of summarizing the patterns within the entire dataset into multiple clusters and the weights associated with each output node represent a prototype pattern.

The GNG algorithm extends the Neural Gas [89] model by enabling continuous learning over time with non-static number of clusters. It borrows the Hebbian competitive learning theories of SOMs to learn a topological representation of the data. A GNG begins its operations by first creating two initial nodes and proceeds with reading the input data. In each iteration, the two closest nodes are identified using distance measurements and an edge is inserted between these nodes. Subsequently, the weight vector of the winning node, which represents the position of the node in the input space, and the weight vectors of its neighbouring nodes are updated to move them towards the input data pointer in proportion to corresponding error values. A

local error parameter for each node keeps track of accumulated error amount computed from the distance measurements. Furthermore, each edge has an innate age parameter which is incremented when the end nodes become winners and is used to track and remove an old edge to keep the topology updated. If the removal of an edge results in an unconnected node, the isolated node is deleted as well. After certain iterations, a new node is inserted between the nodes with the largest error to decrease the total accumulated error and allow dynamic evolution of the topology. The steps are repeated in each iteration until the network converges and becomes stable.

SOINN works similar to GNG with regard to incremental neuron updates and use of edges to evolve the topological structure between nodes [40]. The algorithm uses a two-layer network to learn input features at various levels of granularity by training the second layer only after the first layer has been trained. Another notable difference is the combination of an adaptive threshold for novel class detection and individual decaying neuron learning rate to achieve a more stable representation of the data.

2.3.4 State-of-the-Art ANN Models

The aforementioned models serve as important foundational models for multiple modern stream clustering neural network models. One such algorithm is Adaptive Incremental Neural Gas (AING) [18]. This GNG-inspired incremental model uses a parameter-free adaptive distance threshold to create improved distinction of cluster membership while addressing the issue of network parameter sensitivity. Essentially, the algorithm can produce a compact topological structure using fewer neurons by specifying the distance requirements for creating a new cluster or merging existing ones. A more recent GNG-based network called G-Stream [45] has been proposed. It

operates similar to AING conceptually, with each node in the structure representing a cluster. Notable differences are the use of time-decaying weights, to represent diminishing importance of the weights in evolving data distribution over time, and the design decision to instantiate the model with fewer nodes.

In 2021, a new iteration of GNG was proposed. The GNG-L (L stands for 'Linked List') [91] algorithm improves on the traditional GNG method in three ways: 1) an adaptive coefficient method enables tracking of node movements during concept drift, 2) an adaptive linked list implementation combined with an engineered node removal mechanism allows for feature migration during processing, and 3) a corresponding node creation technique to complement the new linked list method with query capabilities. The resulting network is capable of analysing local topological changes in the data stream and continuously update the feature space to learn the differences.

Many of the SOINN based models are improvements rather than re-implementations. Due to the nature of SOINN's architecture, the neuron weights take longer to stabilize and a high number of parameter values per layer must be learned by the network for optimal performance. The most notable SOINN model improvements include Enhanced Self-Organizing Incremental Neural Network (ESOINN) [41] and Load-Balancing Self-Organizing Incremental Neural Network (LB-SOINN) [132]. ESOINN addresses some of the problems of SOINN by reducing the number of layers from two to one and adding conditions for node insertion. The enhancements reduces the number of required parameters effectively and optimizes the model for online learning, however, it makes ESOINN more sensitive to the sequence of input patterns. As a result, changes in the ordering of the input data greatly affects the topological structure and makes the algorithm unstable. In addition, ESOINN has difficulty in

processing areas of high-density overlapping clusters. It also suffers from reduced confidence when merging and splitting sub-clusters within these areas. LB-SOINN was proposed to address these problems [132]. The model uses node learning time to represent the load on each node and uses the same to balance the network structure. The algorithm supplements high-density clusters with additional neurons to reduce computational time. It also implements a smoothing function to address high-density overlaps and a new similarity measure for calculating distances in online learning tasks.

A set of Bayesian network-based probabilistic generative models have been proposed for classifying human activity data in 2018 [79]. The GPA (generative probabilistic model with Allen's interval-based relations) model and its variations represent the relationships between time intervals for a particular action. Groups (connected nodes) in the network are created when nodes (time intervals) are assembled based on a probability such that they share features of a similar complex activity. Each node is then assigned an atomic action, the specific action within the complex activity sequence. While probabilistic generative models are not true neural networks, we included this work because it trains weight values. In the training phase, GPA attempts to learn the parameters for each node such that their chosen score is maximized.

Recently, a model called DenSOINN (Density Based Self Organizing Incremental Neural Network) [129] has been proposed by combining the architecture of SOINN with the capabilities of a novel density-based clustering algorithm. This network can form clusters of arbitrary shapes with competitive results. The model adopts the online-offline training and testing approach which is typical of streaming clustering algorithms. During the online phase, as each input pattern is presented, competitive

learning is activated among all nodes and the two closest nodes (the winner and the runner up) are identified. If the input data belongs to any of the clusters, connecting pointers are created linking the input, winner, and runner up nodes, and otherwise, a new node is inserted. In the offline phase, the structure is split into connected sub-graphs and the density peaks of each sub-section are designated as cluster centers. Sub-graph elements that are not assigned as center are grouped into the closest cluster with an existing cluster center.

One of the neural network implementations that uses the online-offline phase design is DeepStreamCE [23]. This framework is a combination of two neural networks, a Convolutional Neural Neural (CNN) and an Autoencoder. It applies a stream clustering algorithm called Micro-cluster Continuous Outlier Detector (MCOD) [67]. DeepStreamCE uses the offline phase to train the CNN and the Autoencoder on training data for feature recognition. The trained CNN-Autoencoder returns a reduced set of key features which are passed into the MCOD to create clusters to be used in the online phase. During live clustering in the online phase, the features that are selected by the trained CNN and the Autoencoder are subsequently sorted into clusters by the MCOD.

Another neural network based model that uses an optimally designed variational Autoencoder, has been proposed by [130] for extracting features from incomplete data in wireless multimedia sensor networks. This framework uses high-order fuzzy c-means algorithm (HOFM) to improve the performance of the clustering method. The Variational Autoencoder is a nonlinear method for dimensionality reduction and enables combining probability measures with deep learning [82, 55].

Jakubowski et al [59] use unsupervised learning with a variational Autoencoder

(VAE) to monitor the wear and tear of rolls in a hot strip mill, a part of a steel-making site. There is a lack of labelled data on asset breakdowns which offer a difficult challenge since assets are generally replaced way before they fail for safety reasons. This gives rise to the limited data on run-to-failure cases in a real dataset. The problem is addressed using Autoencoders (AEs) which do not use labelled data for training. Instead AEs try to extract and learn a concise representation of the data, that can be used to efficiently reproduce the near original data. Thus if an AE which is trained offline on the regular or normative data, fails to reproduce the input data during testing phase, it indicates irregularity in the data pattern and helps identify flaw in the process due to possible damage in asset. For this study, data from Hot Rolling Mill (HRM) is used, which is also referred as Hot Strip Mill (HSM) from ArcelorMittal Poland company. The C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) state-of-the-art turbofan engine simulation dataset generated using a C-MAPSS software provided by NASA (National Aeronautics and Space Administration) is also used in the study for validating the Autoencoder models. For the C-MAPSS dataset, the accuracy for AE is 98.1%, and for VAE it is 98.7%. For the HRM dataset, the accuracy for AE is 90.5%, and for VAE it is 90.5%. For the C-MAPSS dataset, the accuracy for VAE is slightly better.

Krajsic et al. [68] use a VAE for anomaly detection in an online process mining environment, which improves the process mining techniques, and in turn, boosts up the business process management. The results are validated against established algorithms like one-class support vector machine (OCSVM), isolation forest (IF), and local outlier factor (LOF). The VAE model is first trained with regular, non-anomalous process data. Then it is applied to process data containing abnormal

data. Excess-Mass and Mass-Volume (EM-MV) based criteria are used to detect anomalous events. The dataset used in this paper is taken from a loan application dataset, containing a collection of artificial event logs of a simple loan application process. The VAE is first trained on this artificial loan event logs. Then the K-means clustering algorithm is applied using the latent space representation from the VAE to create five clusters based on the number of different activities in the observed event log. The approach aims to maximize the EM score and minimize the MV score for which the OCSVM demonstrates the best performance.

Zhao et al [136] use unsupervised ML algorithm based on Bayesian Nonparametric (BNP) and VAE to continually discover new clusters for incoming streaming data. VAE is used for learning the latent representation. This paper uses MNIST dataset, REUTERS-10k, and higher dimensional image dataset STL-10 for validating the model. Normalized Mutual Information (NMI), Adjusted Rand Index (ARI), Homogeneity Score (HS), and V-measure Score (VM) are used to compare the clustering quality of the different models. The proposed model achieves a NMI score of 85.72%, 46.56%, and 75.26% on MNIST, REUTERS-10k, and STL-10 respectively.

WCDS [21] is yet another neural network based streaming data clustering model, which is a weightless model based on a precursor weightless neural network called WiSARD. The authors improve WiSARD by making it capable of clustering streaming data through the integration of the online-offline approach. The model stores and effectively memorizes the binary representation of input patterns using discriminators. With the help of a look-up table storing the least recently used data, the model simulates concept evolution by removing patterns that have not been accessed for a specified period of time. Periodically, an offline agglomerative clustering method is

executed to generate a cluster hierarchy.

Table 2.3: Performances of models discussed in literature review from published data.

Algorithm	Dataset	Metric	Performance
BIRCH [135]	Synthetic clusters Pixel Classification	Variance from known cluster centers Manual Observation	<4% Near Identical
GNG [39]	Synthetic	Topological Formation	N/a
CluStream [1]	KDD-CUP '99 KDD-CUP '98	Alg. Comparison	Magnitudes better Magnitudes better
DenStream [20]	Synthetic Clusters KDD-CUP '99 KDD-CUP '98	Purity	100% >90% >95%
SOINN [40]	Synthetic Clusters AT&T Face Online AT&T Face	Recognition Ratio	100% 90% 86%
ESOINN [41]	Synthetic Clusters AT&T Face + Online Optdigits + Online	Visual Validation Recognition Ratio -	100% (superior over SOINN) 90% + 86% 92.2% + 90.4%
ClusTree [69]	Synthetic Forest Covertype	Purity (depending on level of structure)	95% - 99% 70% - 88%
AING [18]	Pendigit Optdigit MNIST 3 document bag of words	Recognition Rate; V-measure	97.6%; 53% 97.6%; 55% 94.2%; 46.9% 93.7%; 68.5% (across all 3)
LB-SOINN [132]	Artificial AT&T face + Online WebKb R8	Visual Validation Recognition Ratio Accuracy -	Perfect 96.3% + 96.5% 81.2% 89.7%
HA-Stream (I-HAStream) [51, 50]	Synthetic KDD-CUP'99 ICML'04 Forest Covertype	Visual Validation CMM [70]; Purity CMM -	Near Perfect >99%; >95% (N/a; >98%) 100% (98%) >96%
SNCStream (SNCStream+) [12, 13]	KDD-CUP'99 KDD-CUP'98 Electricity Forest Covertype BPaM	CMM	0.95 (0.95) 0.38 0.99 (0.95) 0.96 (0.99) 0.99
G-Stream [45]	KDD-CUP'99	Accuracy NMI Rand	0.99 0.60 0.65
IDEStream [63]	Electricity Consumption	SSE	$8.3E+09 - 1.9E+10$ (better than every other alg.)

2.4 Literature Reviews

This section presents modern literature on the concepts of Autoencoders, LSTM, and Dynamic Learning.

Table 2.4: Continuation of literature model performance.

Algorithm	Dataset	Metric	Performance
WDCS [21]	KDD-CUP'99	V-measure; ARI; NMI	0.2-0.9 (0.6 at end) for all
	Forest Covertype		0.3-0.4 (0.3 at end) for all
GPA [79]	OSUPEL	Accuracy	0.81
	Oppourtunity		0.98
	CAD14		0.98
FISHDBC [32]	Blobs	AMI; ARI	0.98; 0.99
	Synth		1; 1
	Power Consumption		1; 1
	Handwritten Letters		0.96; 0.99
DenSOINN [129]	KDD-CUP'99	ACC; ARI; NMI; RMSE	0.96; 0.86; 0.80; $9.88x10^5$
	HAR		0.60; 0.47; 0.62; 3.62
	Forest Covertype		0.56; 0.02; 0.14; 459.5
	MNIST		0.61; 0.39; 0.57; 1337
	Pendigits		0.69; 0.56; 0.73; 35.32
HOFCM [130]	MNIST	ACC; ARI	85.5%; 0.78
	STL-10		36.44%; N/a
	NUS-WIDE		95.14%; 0.92
AdapVAE[136]	MNIST	NMI	0.857; 0.835; 0.893; 0.894
	REUTERS-10k	ARI	0.454; 0.426; 0.488; 0.454
	STL-10	HS	0.752; 0.702; 0.776; 0.752
		VM	
DeepStreamCE [23]	modified CIFAR-10	Precision	$0.615 - 0.788$
		Recall	$0.414 - 0.565$
		F-Measure	$0.436 - 0.638$
VAE [59]	HRM	Accuracy	90.5%
	C-MAPSS		98.7%
VAE [68]	Loan application dataset	Excess Mass (10^{-3})	6.738
		Mass Volume	1.152
GNG-L[91]	4 static datasets	Accuracy	95.9% - 98.5%
	15 Synthetic evolving streams		81.2% - 98.5%
FIDC [73]	Synthetic	NMI ARI	1.0;1.0
	PenDigits		0.975;0.965
	Waveform		0.601;0.447
	Spambase		0.983;0.992
	Landsat		0.972;0.987
	Image Segmentation		0.928;0.827
	Multi Features		
IMOC-Stream [8]	Forest Covertype	NMI ARI	0.509;0.433
	Sensor		0.723;0.192
	Powersupply		0.466;0.144
	HyperPlan		0.168;0.041

Table 2.5: Metrics used in literature review

Metrics	Equation	Details
Accuracy	$\frac{TP+TN}{TP+FN+TN+FP}$	TP=true positive TN=true negative FN=false negative FP=false positive
Purity	$\frac{1}{N}\sum_{i=1}^{k} max_j \lvert c_i \cap t_j \rvert$	N=total points k=# clusters c_i=a cluster in C t_j=max classification count for c_i
Precision	$\frac{TP}{TP+FP}$	TP=true positive FP=false positive
Recall	$\frac{TP}{TP+FN}$	TP=true positive FN=false negative
F-measure	$\frac{TP}{TP+\frac{1}{2}(FP+FN)}$	TP=true positive FP=false positive FN=false negative
Recognition rate	Correct matches / total matches	Correct total matches in image recognition
V-measure	$\frac{(1+beta)*homogeneity*completeness}{(beta*homogeneity+completeness)}$	homogeneity=if all clusters contains data of single class completeness=if all data points of a class in same cluster beta=weight between homogeneity and completeness
CMM Cluster Mapping Measure	[70]	Metric for clustering evolving data streams
SSE sum of square errors	$\sum_{i=1}^{n}(y_i - f(x_i))^2$	$f(x_i)$=predicted value y_i= value of variable to be predicted
EM-MV Excess Mass-Mass Volume	[68]	Estimates density level curves of probabilitiy distribution

2.4.1 Autoencoder Literature Review

Traditionally, 2D CNNs are applied towards the image recognition domain, since the natural alignment of pixels neatly feeds into the networks. However, some work has been done with 2D CNNs on 1D time-series data, and notably with HAR sequences. For example, Munzer et al [94] published a comprehensive study on CNN-based deep learning methods for HAR and concurrently show the effectiveness of modern implementations for sensor fusion. Another example is the work of Chen et al [28], who applied CNN models onto the MobiAct data and achieved excellent results using their custom dataset structuring process.

In regard to AE models for HAR, the majority of literature apply these neural

networks for data compression and feature dimensionality reduction to improve classification accuracy. The literature written by Almaslukh et al [6] stacked two AE models and combined them with a classifier to create an autoencoder-based classification network and saw comparative improvement of 1.1% in performance. Similarly, the work of Varamin et al [121] presented a deep AE framework which feeds the reduced-latent features into a classification head that predicts a set of activity elements using set inference. In 2019, a stacked denoising AE and gradient-boosted decision tree framework was proposed for HAR [42]. This method was capable of competing with proven techniques such as CNNs in HAR classification. Thakur et al [116] designed a hybrid model consisting of a convolutional AE and LSTM to add temporal processing capabilities when analyzing HAR data. Their approach obtained state-of-the-art accuracies on the UCI HAR dataset.

2.4.2 Long Short-Term Memory Literature Review

The literature review for this chapter will focus on temporal-based deep learning models. A recurring theme in the literature surrounding any kind of temporal learning is multi-sensor fusion techniques - methods that automatically combine sensor information and return a reduced set of features. This method is found in domains that utilizes multiple sensors and requires the need to extract features from all modalities at the same time. As such, the method is very applicable to the domain of HAR [2, 98, 97, 134]. One example is the framework proposed by Bernal et al [14], which concurrently processes video and sensor data to provide greater confidence in model decision-making. The framework relies on a CNN model to learn high-level features and a LSTM model to consolidate and extract the temporal modalities from

the reduced data. The paper applied their model onto the OPPORTUNITY HAR video dataset. Similarly, the work done by Khan et al [64] also uses a CNN-LSTM architecture for temporal feature extraction. The difference is, the authors create their own dataset comprising skeletal joint data of the participants performing the activities. The authors of Challa et al [22] applied the CNN-LSTM architecture on the UCI HAR dataset but opted to increase performance by adding two additional CNN heads. This lead to a powerful multi-headed CNN-LSTM model which achieved high performances (96%) on the dataset.

Some authors have also taken the classic architecture and integrated custom functions or metrics to improve on HAR. One example is the work presented by Cheng et al [29], who proposes a new method for calculating convolutions. They apply a technique called CondConv that is speeds up and improves the performance of convolution calculations. They performed an ablation study and applied a CNN-LSTM model (with CondConv) on the Opportunity dataset and achieved 92% accuracy. Their CNN model (with CondConv) also achieved near perfect scores (99%) on the UCI and WISDM datasets. In the same sense, the authors of Ferrari et al [38] created a deep learning model capable of personalizing HAR signals. By adapting ResNet and proposing a new metric for determining closeness of the subjects ('similarity'), they were capable of creating a model that can distinguish between subjects based on physical similarities and/or sensor value similarities. Their personalized model achieved 90% accuracy on the MobiAct dataset.

2.4.3 Dynamic Learning Literature Review

This section describes some of the recent literature pertaining to dynamic learning. Many of the models listed here are complex architectures typically consisting of multiple components combined to create a hybrid analytical framework, while some of the models are simple independent clustering algorithms. We decided to implement simpler model architectures to gain a better understanding of the clustering approach when applied to complex time-series IoT HAR data, and enable near real-time clustering. A summary table of the described models is presented in Table 2.6.

Semi-Supervised Learning

Semi-supervised learning is a form of machine learning classification that leverages the accessibility of unlabelled data with increased accuracy obtained from using labelled data to build better pattern extraction models [138]. The main issue with using completely labelled data is the effort and time it takes to manually annotate the data. On the other hand, while unlabelled data is easy to collect, it comes at the cost of lower performance since learning algorithms have no objective measure of correctness when classifying data points. Semi-supervised learning models use a majority of unlabelled data and supplement learning with a small number of labelled data to increase classification performance.

The algorithm presented by Li et al [74] is a semi-supervised framework for stream data processing. The framework is made up of several layers, with each layer consisting of what they call a generative network, a discriminant structure, and a bridge to connect the two. The generative networks are denoising autoencoders that learns features from input data. The discriminant structures regularize the autoencoders

by creating pairwise similarities constraints using a hashing function. The bridge enforces the correlation between the parameters of these two modules. The autoencoder incrementally learns features from the input data and reduces the noise in the data by creating a concise feature representation while maintaining reproducibility.

Another semi-supervised approach was proposed by Din et al [33] as a micro-cluster-based model. In the offline training phase, the algorithm creates an initial model from the given data. In the subsequent online phase, the labels for each observation are predicted with a k-NN classifier. Each instance is used to update the learning model incrementally and either added into an existing micro-cluster or used to create a new one. The authors implement a decaying feature called 'reliability' to determine outdated micro-clusters and remain vigilant to concept drift. This algorithm was validated on benchmark stream datasets such as KDD-CUP99' and HAR.

A cluster ensemble approach with semi-supervised learning was presented by Yu et al [131]. The idea behind cluster ensemble is to aggregate results from various clustering algorithms and extracting a single solution from the returned clusters. The authors proposed a new framework that addresses two issues of cluster ensembles: 1) cluster ensembles do not make use of prior knowledge represented by pairwise constraints, and 2) most cluster ensembles do not perform well on high dimensional data. The proposed framework utilizes random subspace, constraints propagation, and normalized cut algorithms to iteratively select the ideal ensemble members based on both a global and local objective function. Datasets include the UCI IRIS dataset as well as multiple handwritten datasets.

Micro-Clustering

The algorithms in this section all implement some form of micro-clustering [20].

The authors of Nguyen et al [96] presented a true stream processing algorithm to address the problem of multi-label classification, where some object can be describe with multiple terms or have multiple meanings. By borrowing the micro-cluster CF from DenStream [20] and combining it with a custom exponential decaying function, the authors enabled incremental clustering with concept drift processing. The algorithm uses label frequency to determine the probability that an input belongs to any of those labels - the top h labels with the largest probability are then assigned to the sample. The algorithm incrementally learns the value of h and adjusts it when it differs from the average number of labels for each sample using a Hoeffding inequality.

MicroTEDAclus is an algorithm that builds on the TEDA framework - a prior design that splits data observations into micro and macro-clusters [85]. This method uses typicalities, a technique for calculating similarity based on proximity, which is determined by eccentricity (a measurement based on mean and variance). This algorithm proceeds in a pure streaming method, with each iteration consisting of two steps: the micro-cluster step and the macro-cluster step. Micro-clusters are determined by the typicality of the incoming data, and if the observation belongs in the space of any micro-clusters, it is added to those clusters; otherwise, it is an outlier and a separate micro-cluster is created. Overlapping micro-clusters results in a macro-cluster.

The paper published by Huang et al [56] presented a new solution that addresses sensitivity to outliers, differing sub-spaces in high dimensional streams, and misjudging outlier points. HEStream is a three stage density-based clustering model that

uses the online-offline approach. By using an Euler kernel function, the distances and similarities of input observations were determined to create micro- and macro-clusters. The model maintains two lists of micro-clusters: the potential micro-clusters and the outlier micro-clusters. Outliers are placed into a buffer and separately evaluated to determine if it 1) belongs to a potential micro-cluster, 2) belongs to an existing micro-cluster, or 3) if it truly is an outlier micro-cluster. In the last stage, the buffer points are relearned and added to the list of potential micro-clusters before the micro-clusters are merged together.

CEDAS (clustering for evolving data streams) is a fully online density-based clustering algorithm that is capable of finding arbitrarily-shaped clusters [57]. The algorithm creates a cluster graph based on overlapping micro-clusters determined by defining a shell and kernel region for each micro-cluster - micro-clusters with a kernel region that intersects another micro-cluster shell region becomes a macro-cluster. Observations that fall in the shell region are assigned to the cluster and the relevant cluster count and cluster center is updated. If the point falls in the kernel then the point is still assigned to the cluster but only the cluster count is updated.

BOCEDAS (Buffer-based online clustering for evolving data streams) is an expansion of the CEDAS algorithm that improves the way CEDAS handles irrelevant outlier micro-clusters [58]. By using a buffer to store temporarily irrelevent micro-clusters, the algorithm is capable of re-accessing these points at a later date to determine usability. The problem with CEDAS is the need for a global and constant micro-cluster radius to be defined - BOCEDAS solves this by updating the micro-cluster radius to its local optima, thereby increasing cluster quality. All other components are largely the same as CEDAS.

Other

The authors of Puschmann et al [105] proposed an adaptive online clustering algorithm capable of adjusting itself to changing trends in the stream. The algorithm analyses the data distribution and updates centroids according to the trend it finds. By using a probability density function (PDF), the authors can determine a change in concept when a directional change in the PDF is detected. A silhouette metric then estimates the best 'k' value to use in a K-means-based clustering algorithm. If the data is not longer converging towards the predicted PDF, the algorithm triggers a re-calibration, dynamically learning new trends as they appear.

A paper published in 2015 [78] proposed an incremental learning algorithm capable of generating a 2D topological representation of the given data. The authors create the dynamic network by first defining smaller hexagonal graphs and constructing nodes onto each corner of the hexagons. The smaller networks are compiled together through competitive learning. If any new data arrives that fails to represent the current distribution, the model splits and saves the distribution into multiple components. When an observation arrives that represents the missing portions of the distribution, the model combines the previously saved components with the new components to incrementally learn the new distribution.

In 2020, Liu et al [81] presented a model capable of multi-class incremental learning. The core concept of the algorithm is passing through exemplars (ideal representations) for each class into subsequent training phases. The authors implemented a method - mnemonics - to automatically extract exemplars from the data stream, parameterize them, then optimize. Using this framework, the model can learn exemplars from each new class while adjusting old exemplars to fit the data distribution.

Their entire framework optimizes two models, the implemented model and the parameterized mnemonic exemplars, and it achieves this by alternating learning between the two models: the classic model is trained on exemplars and and validation loss while the gradients are subsequently back-propagated to optimize the parameters of the mnemonic exemplars.

The paper written by Pari et al [101] proposed the use of ensemble learning to reduce the 'regret' of incremental learning algorithms, which is the difference between incremental and full batch algorithm performances. In order to keep up with the constant information changes, the authors implement the use of ensemble learning (multiple learners) to increase accuracy and adapt to concept drift. A multi-stack of ensemble learners is used and the final generalized prediction is tasked to another classifier.

2.5 Summary

In this chapter we provide a quick summary on the core concepts of data stream clustering. The main theories covered are stream window definitions, data summarization structures, processing stages, and clustering approaches. The implementational achievements in field of streaming clustering are described from an evolutionary perspective. Literature reviews are also provided on the recent work with regard to Autoencoders, Long Short-Term Memory models, and dynamic learning.

Table 2.6: Summary of dynamic learning literature models

Author	Performance	Dataset	Approach	Processing
Nguyen et al (2019)	F-Score: 0.16-0.51	IMDB email collection medical articles	Density	Online
Lie et al (2019)	Accuracy: 75%	MNIST CIFAR-10 IMDB UCI Forest CoverType Youtube Object	semi-supervised denoising autoencoders	Offline-Online
Din et al (2020)	Accuracy: 95%	KDDCup'99 Forest CoverType HAR 7 synthetic	semi-supervised density (w/ micro-clusters)	Offline-Online
Yu et al (2015)	NMI: 0.91	UCI IRIS hand-written 12 high-dimen. cancer	cluster ensembles	Offline-Online
Puschmann et al (2016)	Silhouette: 0.4-0.5	two synthetic stream traffic sensor	probability density w/ K-means	Offline-Online
Liu and Ban (2015)	ARI, NMI, F-measure: 0.99%	static hand-gesture	neural network	Online
Xiaoyu et al (2020)	Accuracy: 25%	CIFAR MiniImageNet	Neural Gas	Offline-Online
Liu et al (2020)	Accuracy: 60%	CIFAR Imagenet	Neural Network	Offline-Online
Pari et al (2018)	Accuracy: 63, 99, 87	KDDCUp'99 Forest CoverType electricity	Cluster ensemble	Online
Maia et al (2020)	ARI: 0.69	static 3 synthetic	Density	Online
Huang et al (2021)	F-measure: 0.817 (HAR) and 0.973 (KDD) NMI: 0.793 (HAR) and 0.967 (KDD)	KDDCup'99 HAR	Density	Online-Offline.
Hyde et al (2017)	Accuracy: 90+%	KDDCup'99 Mackey-glass	Density	Online
Islam et al (2019)	Purity: 90+ Accuracy: 90+%	KDDCup'99 weather data	Density	Online

Chapter 3

Examining Feasibility and Efficacy of Traditional Stream Clustering Algorithms

This chapter contains the first steps in our exploration of machine learning clustering models. Here, we present several standalone clustering algorithms and evaluate their performances on streaming HAR data. This serves as a fundamental stepping-stone upon which to build the rest of our research where we explore the established statistical clustering algorithms designed for both streaming and non-streaming data. This experimental study mainly focuses on addressing the question: are established statistical models capable of extracting HAR domain specific knowledge such that clear clusters are returned? Most of the literature on clustering does not overlap with HAR - most of the datasets use famous benchmark datasets which are no longer sufficient in determining efficacy of these algorithms for other data domains. As such, we apply the algorithms onto complex streaming HAR data to analyze the limits and establish the baseline for HAR clustering performance.

The motivation behind the experimentation presented in this chapter is to gain an understanding and test the limitations of popular clustering models in clustering

streaming sensor data for HAR. While there is rich literature on clustering streaming data, little work has been done on applying clustering methods on streaming sensor data from Inertial Measurement Units (IMU) containing accelerometer, magnetometer and gyroscope data for HAR [32, 129, 79]. Our prior work focuses on applying supervised learning methods for HAR on labeled IMU sensor data [28, 84, 4]. In this work we aim to explore existing stream data clustering methods for a complex time series IMU dataset for HAR. Each reviewed paper in this work presents either a generalized method applied to a benchmark dataset, or a specialized model engineered for a particular type of data. One problem is the wide utilization of the inappropriate KDD-CUP'99 dataset as a benchmark for clustering streaming data. Wang et al [123] and Tavallaee et al [115] explored this popular dataset and determined it to be too simple, with even the most fundamental algorithms achieving high performances. Presumably, this issue of noncompetitive benchmarks extends to other datasets (such as MNIST [11]). This implies that the superior performances listed by the authors could be attributed to simplistic data rather than efficient algorithm design. Specialized models designed for specific problems clearly perform well when applied to the intended datasets, but their adaptability to other problems must be tested by using other test datasets. For example, Chen et al [28] applied a model that is conventionally applied to image data (CNN) to HAR data and achieved excellent classification results. By applying some of the generalized clustering algorithms to an intricate dataset such as the sensor IoT data for HAR, we can test the general competency of the algorithms and validate their scalability and adaptability for the HAR application domain. At the same time, the study provides insights about the architectural design, functionality and performance of the algorithms to allow the

selection of the best algorithms for real time HAR based on time series IMU sensor data.

3.1 Selection of Clustering Algorithms

The algorithms we pick for this chapter all incorporate different clustering concepts to expand our possibilities for later implementations and to cover a larger base. For example, perhaps a density-based clustering algorithm will have trouble discerning overlapping actions while an algorithm that utilizes a different distance metric could tell the sequences apart. Our justifications for choosing each algorithm come from literature reported performance on either similar or seemingly more complex data. All chosen models obtained high ARI and NMI scores (0.8-0.9) on the benchmark data streams described above. Hierarchical Density-Based Clustering (FISHDBC) [32] which is flexible,incremental, and scalable, is built off DBSCAN [37] and HDBSCAN [19] and as the name implies, is a density-based clustering algorithm. Its predecessor uses a Minimum Spanning Tree (MST) to represent the distances of each point to its closest neighbour. FISHDBC improves on this mechanism by estimating the MST instead of hard-computing all the distances, thereby increasing computational speed. Additional improvements allow the algorithm to update the MST incrementally rather than fully constructing it at each new call. WiSARD for Clustering Data Streams (WCDS) [21] is an incremental version of the classic WiSARD weightless neural network [5]. Being weightless, WCDS is not constructed like other neural networks, with their weights matrices, instead the model uses RAM inspired data structures called discriminators which store and memorize a binary representation of input patterns. Finally, Deep Embedded Clustering (DEC) [127] is an Autoencoder

(AE) model with a custom clustering layer in place of a classification head. This model leverages the power of deep neural networks to break down and learn features within the stream by translating data features into a low dimensional space and optimizing the Kullback-Leibler (KL) [71] Divergence loss. With these three algorithms, we evaluate their adaptive capabilities on complex HAR data and consider their use in later experiments.

3.1.1 Outcomes

The experimental results demonstrate that both WCDS and FISHDBC are able to achieve near-identical Normalized Mutual Information (NMI) scores and relatively close Adjusted Rand Index (ARI) scores when clustering the data streams (0.559 ± 0.008 for WCDS, and 0.42 ± 0.07 for FISHDBC respectively). The AE-based DEC model demonstrates similar but slightly worse results compared to the other algorithms, achieving clustering scores of 0.498 NMI and 0.403 ARI. Our conclusion from these results is that despite the reported capabilities of the literature models, they are incapable of effectively clustering the MobiAct HAR data because 1) the MobiAct data is far more complex than the benchmark datasets used in literature, and 2) the model architectures are lacking the analytical depth required for meaningful processing. We believe that supplementing the clustering algorithms with other models capable of feature extraction/reduction is required to produce better and distinct clusters.

The rest of the chapter is organized as follows. Section 3.2 provides an overview of the HAR problem using unlabeled complex streaming IoT sensor data and describes relevant datasets. The research methodology including the models selected to be

implemented and validated with HAR data are described in Section 3.3. Implementation and data pre-processing are illustrated in Section 3.4. Section 3.5 presents the validation results and discussions. Finally, Section 3.6 concludes the chapter and lists the next step in the exploration process.

3.2 Clustering Sensor Data for HAR

3.2.1 Problem Description

Human Activity Recognition (HAR) is a complex problem because of the intricacies that exist in human movements. Machine learning models that rely on pattern recognition must be robust enough to disregard the slight variances with the same movements while recognizing the boundaries between transitioning actions. Additionally, this problem is made more complicated when the data come in a streaming form. As such, the criteria for selecting algorithms to validate in this study include accuracy, computational cost, and efficiency. Existing literature on stream clustering is highly saturated with algorithms engineered for an encompassing selection of problems. They are typically presented as scalable and adaptive, which are capable of processing a variety of data types with satisfactory performance. We take several of these algorithms and apply them to complex streaming HAR time-series data. We evaluate the efficacy of these methods and consider their usefulness for future work.

3.2.2 HAR Datasets

The HAR datasets typically consist of multidimensional time-series data that are collected using a variety of sensors, where each row of data is labeled with a type of activity (as shown in Fig 3.1). Modern smartphones and smartwatches have the

hardware, software capabilities, and internet connectivity to enable continuous data recording. Smartphones are one of the preferred methods for HAR data acquisition. According to the literature, some of the popular benchmark datasets that are frequently used by the researchers are the UCI [7] dataset collected using smartphones, the Wireless Sensor Data Mining dataset (WISDM) [72], the Opportunity challenge dataset [26], the USC human activity dataset (USC-HAD) [133], and the MobiAct v2.0 HAR dataset [25] from the Biomedical Informatics and eHealth Laboratory. The UCI dataset consists of ten-thousand samples pertaining to only six classes of activities, but includes additional sensory information on postural transitions i.e., the motions encountered when moving from one body posture to another. The 2012 WISDM dataset contains solely the accelerometer readings of six activities over five thousand data samples. When introduced initially, it was considered as a benchmark dataset. The Opportunity challenge dataset stood out from the rest as it includes magnetometer readings in addition to accelerometer and orientation information. Another dataset that has been released in 2012 is the USC-HAD dataset, a comparably small dataset (840 samples from 14 individuals) but it represents 12 different activities rather than the typical six activities as presented by the other datasets. Considering the complexity, sample sizes, and activity counts, we chose to use the MobiAct dataset in our experiments for this Chapter. Further details of the dataset are explained below.

Table 3.1 shows a summary of the different HAR datasets that were discussed earlier.

Table 3.1: Summary of the HAR datasets.

Dataset	No. of Subjects (Female/Male)	No. of Activities
MobiAct v2.0 [25]	57 (15/42)	16
USC-HAD [133]	14 (7/7)	12
Opportunity challenge [26]	4	4
WISDM [72]	36	6
UCI HAR Using Smartphones [7]	30	6

3.2.3 Dataset for Experiments

To explore the hierarchical patterns for HAR, we require streaming data representing continuous activities, and demonstrating transition of movements and correlation between actions. Therefore, we opted to use the scenarios of daily living which demonstrate the above, instead of disjoint individual activities.

The 2017 MobiAct v2.0 HAR dataset improves on the previous iteration by including data from falls and additional classes of activities. It represents 16 activities of daily living, which include common actions such as walking and standing; activities that are fall-like; and activities with sudden movements similar to fall (jumping; jogging). The new iteration, also, includes scenarios of daily living, which are a series of activities that make up common everyday routines (leaving home; coming home from work etc.). The scenarios consist of 3200 trials from 66 subjects with a varying number of activities. Each activity was recorded using a smartphone with a 3D accelerometer and a gyroscope, in addition to orientation data.

Below is an example of the MobiAct dataset (Fig 3.1). Each instance of the data contains 12 columns. The *timestamp* column reflects the time and date as recorded by the Android API. The *rel_time* column is the time of that observation within the trial. The 9 features contain the readings of the axis for each sensor. The last column is the abbreviated label of the action in that observation. The activity labels were assigned manually by the researchers.

timestam	rel_time	acc_x	acc_y	acc_z	gyro_x	gyro_y	gyro_z	azimuth	pitch	roll	label
3.42E+12	0	-0.47244	9.843034	-0.49866	-0.02291	0.042455	0.011606	313.4496	-48.5263	-3.21032	STD

Figure 3.1: An instance of the MobiAct dataset

For our model development and validation, we choose a specific daily living activity, which is the 'Scenario of Leaving Home' (Table 3.2), because it represents a sequence of actions most similar to real-life scenarios due to the natural flow of movements into each other. This sequence comprises six general classes with nine distinct instances for around 23,900 data points. This sub-dataset is used for all the models. The entire MobiAct collection of activities is also given in Tables 3.3 and 3.4 for reference, where Table 3.3 contains all the non-fall activities and Table 3.4 lists the fall actions.

While we will not be using this dataset in this chapter, the UCI HAR dataset will be consistently utilized in the rest of the thesis, so we will describe the dataset now alongside the MobiAct collection. The UCI HAR is a 2012 dataset [7] donated to the University of California Irvine Machine Learning Repository. The dataset comprise of 10299 instances (time sequences) split between 30 subjects between 19-48 years old. It only has 6 activities: walking, walking upstairs, walking downstairs, sitting, standing, and lying. A pre-processed version is supplied by the author, but for the purpose of our experiments, we will be using the raw data sequences. Note that by

Table 3.2: The 'Scenario of Leaving Home' activity labels and the order in which they appear in the dataset.

Label	Activity	Description
STD	Standing	Scenario Leaving Home (SLH): The agent stands outside a door and locks it. The agent then descends a flight of stairs to the parking area where they stop in front their car. The agent unlocks the door, opens it, and steps into the car. They sit for some time before stepping out of the car. The agent closes the door and stands still while locking the car.
WAL	Walking	
STN	Stairs Down	
WAL	Walking	
STD	Standing	
CSI	Car Step In	
SIT	Sitting	
CSO	Car Sep Out	
STD	Standing	

'raw' data, we mean the noise-reduced and sampled version of the inertial data, and not the raw sensor values directly harvested from the phone, as those values are not available to us. The 3 axis of the two phone sensor (accelerometer and gyroscope) are used to construct the data values, but an additional set of features is manufactured for a total of 9 feature columns. Just like the MobiAct data, the UCI classes are manually labelled via video by the researchers. One caveat of the data is the that the sequences come sampled using fixed-length windows of 2.56 seconds and 50% overlap, ultimately creating sequences with 128 readings per window. The shape of each sequence is (1, 128, 9).

Table 3.3: MobiAct v2.0 activities.

Label	Activity	Duration	Description
STD	Standing	5min	Standing with minimal movements
WAL	Walking	5min	Typical walking
JOG	Jogging	30s	Jogging
JUM	Jumping	30s	Continuous jumping
STU	Stairs Up	10s	Going upstairs (10 steps)
STN	Stairs Down	10s	Going downstairs (10 steps)
STN	Stand to Sit	6s	Transition from stand to sit on chair
SIT	Sitting	1min	Sitting on chair with minimal movements
CHU	Sit to Stand	6s	Transition from sit to stand from chair
CSI	Car Step In	6s	Stepping in a car
CSO	Car Step Out	6s	Stepping out of car
LYI	Lying	–	Lying down after fall

Table 3.4: MobiAct v2.0 falls.

Label	Activity	Duration	Description
FOL	Forward-lying	10s	Falling forward from standing, landing on hands
FKL	Front-knees-lying	10s	Falling forward from standing, landing on knees
BSC	Back-sitting-chair	10s	Falling backwards while trying to sit on chair
SDL	Sideward-lying	10s	Falling sideways from standing with legs bent

3.3 Materials and Methods

We presented a literature review of statistical and artificial neural network models designed for clustering in section 2.3. To validate the applicability and efficiency of these models when used with time series IoT sensor data for HAR, we selected two statistical methods namely, FISHDBC and WCDS, and one neural network model, DEC. In this section, we justify the selection of these models and then describe the

model architecture and implementation including the methodology and parameter choices. Next we present the data pre-processing and experimental details. The validation results are presented in the next section.

3.3.1 FISHDBC

The FISHDBC [32] model is an extension of HDBSCAN, and the underlying method is density-based clustering. While traditionally the HDBSCAN algorithm creates a new Minimum Spanning Tree (MST) with every data update, FISHDBC is capable of greatly improving the computational complexity by constructing an approximation of the MST and at the same time maintaining cluster quality [32]. This model approximates the distances between pairs of known data points. If the distance between a pair of nodes is approximated to be computationally insignificant, the model assigns the distance of the pair as $d(a, b) = \infty$, so the algorithm knows to skip the computation for those points. FISHDBC also concurrently maintains a nearest neighbour approximation using Hierarchical Navigable Small Worlds (HNSW) [87], a technique which creates a layered structure of approximated k-nearest neighbour graphs. Each layer of the structure contains $1/k$-th nodes of the layer below it, and the bottom layer represents the entire dataset. FISHDBC stores a summarization of the HNSW structure as it builds and independently queries this smaller structure to calculate the distances between nodes, which improves computational speeds in enabling incremental updates effectively.

FISHDBC takes in several parameters that are used by HDBSCAN and HNSW, and a distance parameter that is used by FISHDBC itself. Other optional but notable parameters include *min_samples* (which affects how conservative the model is

- the lower the value, the fewer points are labeled as noise), *min_cluster_size* (which determines the minimum size of a cluster), and *ef* (which influences the effort HNSW puts in). An additional parameter is *cluster_selection_epsilon* - this determines the distance threshold when a cluster splits into smaller clusters.

3.3.2 WCDS

The second model is the streaming implementation of the WiSARD weightless neural network called WCDS [21]. As the name implies, these types of neural networks do not use weight matrices, instead they implement discriminators for the learning mechanism. Each discriminator (Δ) is responsible for memorizing the patterns of a single class ($\Delta_{\dot{y}}$). The discriminators are comprised of an array of σ set-like nodes, whose responsibility is to split the incoming data into smaller portions. The features of each class are converted into a binary encoding and stored as addresses in the discriminators. Two model parameters, β and γ, control the length of the addresses and encoding resolution respectively. The stored addresses are accessed when an input pattern is received and compared for their pertinence against the new binary encoding (Eq. 3.1). Ultimately, the chosen class for the input observation is the discriminator that results in the greatest match (Eq. 3.2).

$$\text{matching}(\dot{y}, x) = \frac{1}{\sigma} \sum_{i} [\text{addressing}_i(x) \in \Delta_{\dot{y},i}]^1 \tag{3.1}$$

$$\hat{y} = \underset{\dot{y}}{\text{argmax}}\ \text{matching}(\dot{y}, x) \tag{3.2}$$

The addressing function in Eq. 3.2 is presented as a composite function of (if considering the feature space $\mathbb{R}^n$) $g \circ f : \mathbb{R}^n \to \{0,1\}^{\sigma \times \beta}$, where $f : \mathbb{R}^n \to \{0,1\}^{n \times \gamma}$ is

the encoding function that returns the binary representation of an observation, and $g : \{0,1\}^{n\times\gamma} \rightarrow \{0,1\}^{\sigma\times\beta}$ is a random mapping defined prior to any training.

WCDS uses the same general structure as the classic WiSARD system, with the core differences manifesting in the design decisions that the authors implemented to support stream processing. The first change is a last recently used (LRU) lookup table that lists discriminators in the order of last activations. A timestamp is embedded in every entry for temporal analysis. As the LRU table fills up, the lowest entry and the corresponding discriminator is deleted to remove obsolete knowledge representing unlearning over time. The second change deals with cluster imbalance. Since WiSARD learns by memorizing data patterns, clustering results can be heavily skewed by a single discriminator dominating all incoming data observations simply by memorizing the greatest amount of features. The authors propose a solution by normalizing the value that is returned by the matching function. Consequently, as the number of addresses in a discriminator increases, the final value that is returned by the function decreases, and therefore, the impact for that discriminator is effectively weighted.

WCDS also takes in several parameters, some of which have already been mentioned. δ is the resolution of matching between clusters and the input observations that are performed by WCDS; γ determines the resolution of the binary representations of input observations; β controls how strict the similarity is between binary features in matching computations; μ controls the tendency for defining a small number of clusters covering a large area or many clusters covering a small area; and ω is the length of the time window before a discriminator is considered expired.

3.3.3 DEC

The third and final model illustrated in this chapter is an integrated Autoencoder (AE) architecture with a custom clustering layer first proposed by [127]. Deep Embedded Clustering (DEC) combines a deep learning technique called Deep Embedded Learning that can learn features of the data and cluster centers by transforming the data into a lower dimensional space and iteratively optimizing the Kullback-Leibler (KL) [71] Divergence loss (Eq. 3.3). The KL Divergence loss indicates how much information is loss when approximating different distributions. The idea behind the score is that if the probability of an event in distribution P is large but it is small in distribution Q, then the function returns a large score - a large divergence. However, if the probability of P is small and the probability of Q is large, then it indicates that a large divergence still exists but the value that is returned is smaller than that in the other scenarios [71].

$$KL(P||Q) = \sum_{x \in X} P(x) * \log(\frac{P(X)}{Q(X)}) \tag{3.3}$$

For our implementation, the data is first passed through the encoder part of the deep AE to retrieve the reduced embedded feature representation. Then it is passed into a K-means clustering algorithm to determine the centroids for the new clusters. Updates on the clusters and the corresponding centroids are performed using the soft assignment method by implementing student's t-distribution as a kernel to measure the similarity of the neighbouring data points with the centroids. The architecture of the DEC model is provided in Fig 3.2.

The AE architecture consists of simply stacked dense layers. A notable feature is the vast number of nodes at each level. The encoder portion of the AE has 3

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 784)]	0
encoder_0 (Dense)	(None, 500)	392500
encoder_1 (Dense)	(None, 500)	250500
encoder_2 (Dense)	(None, 2000)	1002000
encoder_3 (Dense)	(None, 10)	20010
clustering (ClusteringLayer)	(None, 10)	100

Total params: 1,665,110
Trainable params: 1,665,110
Non-trainable params: 0

Figure 3.2: Architecture of DEC model.

layers with 500, 500, and 2000 nodes respectively. The decoder is the inverse of the encoder portion and the encoded representation layer (located between the encoder and decoder) reduces the data features to a mere 10. The model is pre-trained for a certain number epochs to instantiate the model parameters, then it is further trained until a stopping criterion is met: either 20,000 training iterations have passed, or the delta (change in loss) is below a tolerance threshold.

3.4 Implementation

Two independent stream processing algorithms and one AE model are explored and validated through experiments. All models process the 'Scenario of Leaving Home' sub-dataset from the MobiAct dataset to test their abilities toward true live-stream clustering. All codes are written in Python3 and ran on a Microsoft Windows 10

operating system with an Intel Core i7 6700HQ processor and 8GB of DDR4 RAM. The AE model is pre-trained for 300 epochs using the *Adam* optimizer and *Stochastic Gradient Descent (SGD)* as the loss function. The hyper-parameters for each model were initially setup based on literature and later adjusted depending on empirical results.

3.4.1 Data Pre-proccessing

Data pre-processing is necessary as the MobiAct data is quite complex. While the authors of the dataset [25] already removed the outliers and presented a clean set of data, the intricacies of human movements still prove to be a bit challenging for the algorithms if prior processing is not performed.

For FISHDBC, we apply *MinMax* normalization to reduce the range of distance for the model to analyze. The values in Eq. 3.4 are measured in Euclidean distance.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{3.4}$$

The *timestamp* column has been dropped as the values are too complicated and unnecessary, since we had another time column (Fig. 3.1). The *rel_time* column has been dropped as well, since the models did not use temporal data. The *label* column is unused during clustering and only used after the clusters are defined to compare results. Data is fed to the input nodes using a sliding window with no overlap.

For WCDS pre-processing, we use *MinMax normalization* (Eq. 3.4) - in fact, it was required for the data range to be between -1 and 1 for the model to work. The *timestamp* and *rel_time* columns have been dropped as temporal information was not needed and the *label* column was kept but not considered until cluster evaluation.

The data was fed in sequentially.

For DEC, *timestamp*, *rel_time*, and *label* columns have also been dropped, just like the previous two models. The data is also normalized to be between 0 and 1 for easier processing.

3.5 Validation

In this section, we present the validation metrics, results and a critical discussion of the observed model performances.

3.5.1 Validation Metrics

Normalized Mutual Information Score (NMI) and Adjusted Random Index Score (ARI) determine the validity of the returned clusters. All the models use ARI and NMI for validation.

NMI (Eq. 3.5) is a symmetric measure for the degree of dependency between clustering and true classification. NMI values close to 1 indicate high similarity between clustering and found labels, while values close to 0 indicate high dissimilarity.

$$\text{NMI(Y,C)} = \frac{2 \times I(Y;C)}{[H(Y) + H(C)]} \tag{3.5}$$

where Y are class labels, C are cluster labels, $H(.)$ is the respective entropy, and $I(Y;C)$ is the mutual information between Y and C. Mutual information is given as $I(Y;C) = H(Y) - H(Y|C)$; $H(Y|C)$ is the entropy of class labels within each cluster.

Rand Index (RI) computes the similarity between clustered data and labeled data. The ARI (Eq. 3.6) is ensured to return a value close to 0 for random labeling and 1

when clustering is perfect.

$$\text{ARI} = \frac{(RI - Expected\_RI)}{(\max(RI) - Expected\_RI)} \tag{3.6}$$

where $RI = \frac{a+b}{a+b+c+d} = \frac{a+b}{(n(n-1)/2)}$, where given the dataset S and two partitions of S, $X = \{X_1, ..., X_r\}$ with r subsets, and $Y = \{Y_1, ..., Y_s\}$ with s subsets:

- a is the number of element pairs that are in the same subset in X and the same subset in Y
- b is the number of element pairs that are in different subsets in X and in different subsets in Y
- c is the number of element pairs that are in the same subset in X and in different subsets in Y
- d is the number of element pairs that are in different subsets in X and in the same subset in Y

3.5.2 Results

The performance results for the three algorithms are listed in Table 3.5. Figures 3.3 and 3.4 provide the performance distribution of the WCDS and FISHDBC algorithms respectively.

Table 3.5: NMI and ARI scores for the algorithms.

Algorithm	**ARI Score**	**NMI Score**
WCDS	0.496	0.567
FISHDBC	0.347	0.551
DEC	0.406	0.498

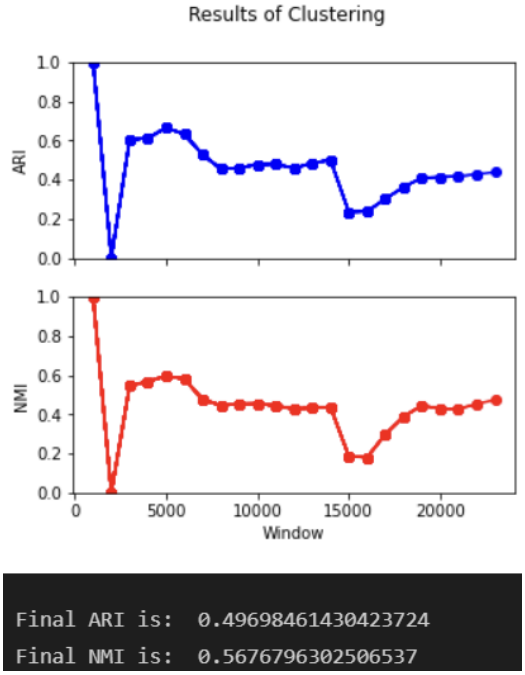

Figure 3.3: WCDS performance distribution

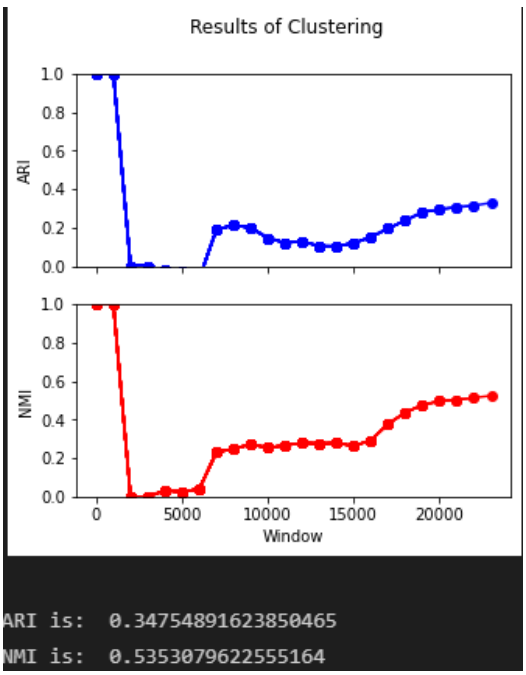

Figure 3.4: FISHDBC performance distribution

3.5.3 Discussion

One of the problems which might have caused subpar results has to do with the data itself - the MobiAct data has dissimilar sensor readings. An example of this can be seen in the first and last thousand points of 'Scenario Leaving Home'. Both sequences are labeled as *STD* but while some values under the same features are similar enough, others are vastly different. This greatly affects how the models perceive the data and potentially mislabel. While the effects can be remedied by using pre-processing scaling techniques such as normalization or min-max scaling, a subsequent issue of homogeneous readings belonging to different classes appears. In addition, transitions from one activity to another can occur with subtle shifts in data values, making it difficult for the models to determine whether the change is simply a part of the action's

range of motions or the beginning of another movement. The ability to differentiate these boundary points greatly influences algorithm performance.

WCDS algorithm works by memorizing a binary representation of the data and groups data points into clusters with the greatest binary matching. This, however, can be difficult because of the data intricacies mentioned above. Since the sensor information for one activity can vary quite a bit, the algorithm has to match a larger range of binary information. To help assist with the range of processing, the dataset is scaled using a minmaxscaler, so all values reside between 0 and 1. However, the large subset of overlapping points can easily cause inappropriate clustering especially if a data point falls in the binary range of one of the larger activities such as *WAL*. For example, a thousand points of *WAL* create discriminators memorizing vast patterns for *WAL*, with none yet created for *STN*. When the data points for *STN* arrive, the readings are similar enough to *WAL* so that the information are most likely grouped with one of the *WAL* clusters. The subsequent *STN* readings are not distinct enough to escape from the range of the *WAL* discriminators and the majority of the activity is clustered together with another. Referencing the performance distribution graph shown in Fig 3.3, we can see the initial perfect clustering of the *STD* activity before the model experiences a drop in performance at the arrival of the *WAL* activities. The poor ARI can be explained by WCDS creating multiple discriminators to memorize the intricacies of the walking movement. The activity is, however, complex and each shift in sensor is significant enough to register the creation of a new discriminator rather than the inclusion of the data into an established one. The results of the model are clearly on a rising trend as the scenario ends; this indicates that the performance can increase if more data points are processed. Potential future work may consist

of clustering the entirety of the MobiAct dataset rather than a single scenario, a method that is reminiscent of classical methods that are used in typical neural network training phases.

For FISHDBC an issue specific to the model is its tendency to misclassify points into the densest, overlapping class. The *WAL* activity has dominated the dataset by a large margin. As such, the model is heavily biased towards the activity, since FISHDBC clusters are based on density. At the same time, because this class is so dominant, the activity is prone to variances in sensor values. The problem here is that these varying points still maintain the same label. This increases the chance for the model to mislabel data as noise or as belonging to surrounding classes. In addition, large instances typically mean greater variance in sensor values in general. This affects the model by causing multiple classes to be identified for those sequence of points. FISHDBC could have difficulty in differentiating between the boundaries of the starting point of one activity and the ending point of another activity. The algorithm has, however, been able to maintain decent performance, as shown in Fig 3.4. The cluster quality drops temporarily, as seen in the dip of the second point, where this point represents the activity that has been transitioned from *STD* to *WAL*. Subsequent dips are the inclusion of other activities in the data stream and presumably the creation of another cluster center. We also notice that clustering performance slowly increases as more points as read in. Since FISHDBC is a density-based algorithm, the model requires a large volume of points to create the density-based clusters - if certain points are not close to enough to other points, they are clustered as noise. Therefore, as more points arrive, the better the algorithm can determine areas of high or low density.

In reference to the performance distributions together (Fig. 3.3 and Fig. 3.4), we can observe a similar pattern in the initial updates for both algorithms. Both models maintained perfect clusters for the singular activity (*STD*) until the arrival of the first *WAL* point, which upon clustering, immediately dropped the scores to zero. The scores at each window are representative of the clustering quality of all points up until that window - meaning, at the inclusion of the *WAL* activity and given the clustering scores of zero, the algorithms randomly assigned points into clusters because presumably the *WAL* data overlapped too much with the *STD* data. Intuitively, this makes sense because the *WAL* data points in this context are transitional points from *STD*. Since the models do not have labels to reference, the transitional points were either clustered together with *STD* or caused the previously established *STD* cluster to break apart into smaller clusters, explaining the drop in performance. Another observation regarding the two algorithms is the capability for them to recreate the performance distribution consistently. While the exact performance scores may vary slightly, the shape and trends in the distributions are observed with every iteration. This shows that the clustering models are capable of capturing the obvious patterns within the data, but their architectures are not complex enough or the data stream is too complex to extract any deeper patterns in the data.

The scores observed by the DEC model are not surprising given the relative performance of the previous two models and falls within the overall range of the scores returned by FISHDBC and WCDS. If we just look at the NMI metric, we could make a case that DEC actually performs worse, since 0.498 is roughly a 10+% decrease from the 0.56 average of the other two. The ARI score fluctuates, so a clear baseline

cannot be determined. Despite being a powerful neural network, the DEC architecture has trouble processing the HAR data. We do know that the model is capable of reducing some knowledge based on the reported loss values (Fig. 3.5), but we assert that further feature reduction is necessary to obtain better clustering results. Several pieces of literature achieved excellent performance on HAR data using just the accelerometer and gyroscope values [28, 7, 22], and since the MobiAct data also includes orientation data, the model should be capable of compressing the data in a way that excludes most of the orientation values. The quick drop in loss value and eventual stagnation seems to imply DEC did easily reduce the feature set (perhaps by removing some orientation knowledge) in the first couple epochs but found difficulty in further feature compression afterwards. The model was able to quickly reduce the loss from 0.0359 to around 1.9e-04 and the clustering quality from the initial 0.357 NMI to a 0.493 NMI in just a few epochs. However, the rest of the training stage saw the loss stagnate and the clustering metric remain mostly unchanged. In comparison, DEC when applied to the MNIST dataset had gradual but continuous improvement in both loss and clustering results [128]. We can conclude two things: 1) the high performance of DEC on the MNIST dataset compared to the relatively low performance on the MobiAct data shows the higher complexity inherent residing in HAR sequence data, and 2) the MobiAct data is capable of being compressed and reduced but requires additional methods of processing to further extract the hidden knowledge within the features.

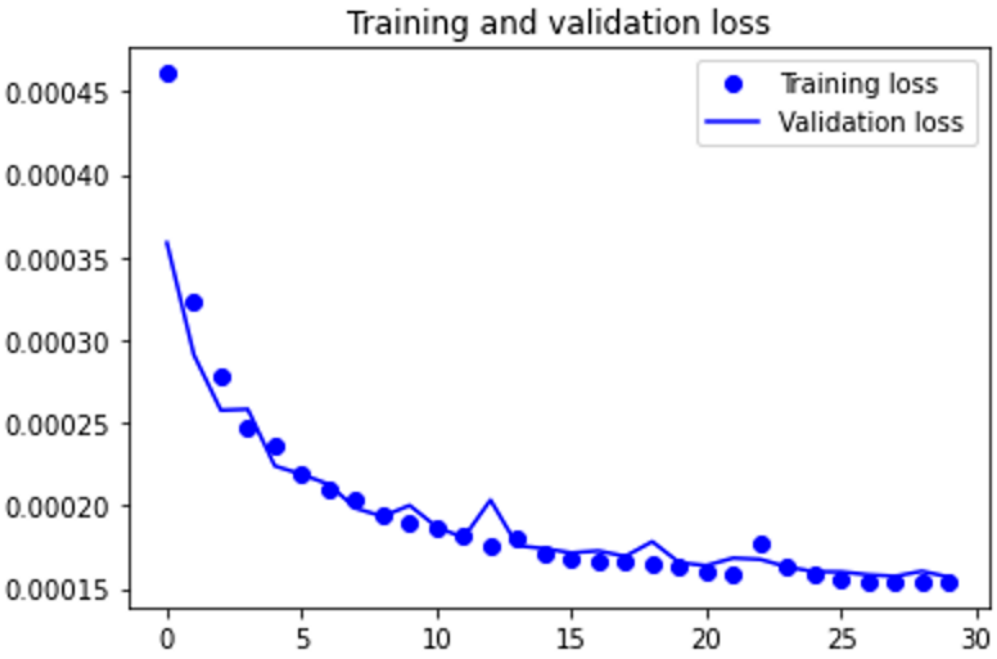

Figure 3.5: DEC loss during training

3.6 Summary

This chapter highlights the lack of use of HAR data in model implementation and validation. We also argue why certain benchmark datasets such as MNIST and KDD-CUP99' are no longer appropriate to assess applicability and efficacy of models to other data domains, and have inflated the perceived performance in literature algorithms. We present an investigation into the effectiveness of three established and reputed stream data clustering algorithms from the literature namely the FISHDBC (statistical), WCDS (weightless neural network), and DEC (deep learning AE) on HAR data. We implemented the models using the public MobiAct v2.0 streaming IoT sensor data for HAR. A comparative analysis of the results reveal that the algorithms generally perform poorly for abrupt variations in the data such as quick change

in posture of the body or elevation. The single statistical model, FISHDBC, adopts and improves on HDBSCAN's hierarchical density-based approach by approximating the nearest neighbour search graph. This method achieved ARI and NMI scores of 0.347 and 0.551 respectively. WCDS, the weightless neural network, uses binary memorization instead of feature weights to learn patterns within the data, and achieved 0.496 ARI and 0.567 NMI scores respectively. Finally, the DEC AE model leverages the power of stacked neural layers to reduce and compress data features to return an ARI score of 0.406 and a NMI score of 0.498. From these results, we conclude that given the complexities and intricacies within the data, unless further processing or an inclusion of temporal extraction is present within the analytical pipeline, these models are incapable of returning distinct clusters on HAR data streams. In the next two chapters we explore ways to improve the clustering models, first by improving the AE architecture and then by extracting and incorporating temporal features in the clustering model.

Chapter 4

Autoencoder based Clustering Models

Although our Autoencoders (AE) based DEC model did not perform as well as the WCDS and the FISHDBC model, autoencoders are commonly applied to reduce noise and extract key features from multi-sensor IoT data in developing predictive models in multiple application domains [103, 47, 10], Therefore, we wanted to further explore autoencoder-based clustering models. By implementing more powerful AE neural networks to forcibly learn the intricacies of the HAR sequences and extract the compressed knowledge, and using those learned features to build clustering models, the quality of clustered may be improved. Also instead of using integrated frameworks like DEC, we propose several different AE architecture to extract a latent representation containing the core features, which can be independently trained and used with classification or clustering backends.

Before experimenting with the different AE models, we revisit the DEC model and train it on the full MobiAct dataset to determine if the poor prior performance was due to a lack of sufficient data points. Next we explore several AE architectures are explored based on the literature such as the CNN AE, LSTM AE, and a hybrid CNN-LSTM AE to determine the best performing model for HAR datasets. Concise

features from the AE are used to train a clustering model.

This chapter will progress as follows: Sections 4.1.1, 4.1.2, 4.1.3, and 4.1.4 respectively presents the implementation details and experimental setup of the DEC model using full MobiAct data, 2D-CNN-AE, LSTM-AE, and the hybrid CNN-AE + LSTM framework. Section 4.2 provides the experimental results and outlines specific comments regarding the corresponding AE model. Before concluding this chapter with a summary in Section 4.6, we present an analysis of the experimental findings, observations, and results in Section 4.5.

4.1 Implementation

The majority of our implementations uses a two step approach somewhere in the analytical pipeline - the data is reduced and compressed via the AEs, then the latent features are extracted and fed into a clustering algorithm to generate clusters - the main differences between implementations is which step this occurs in. The main motivation for AE use is dimensionality reduction and feature compression, which ultimately result faster processing time and ease of analysis. For validation, we use classification to form the basis of performance, since we can use accuracy as an object measure of efficacy for the extracted features. Additionally, the supervised learning approach has access to the true labels, therefore we know the results are not arbitrary. The first approach, DEC, has the AE layers built into the model and uses a custom clustering head to create the final results. The second method initially uses a supervised 2D CNN to learn from the HAR data. The learned features are then extracted and passed into an AE model for compression and then into a simple classifier for evaluation. Next, we account for temporal dependencies by creating a LSTM AE.

The temporal features are subsequently analyzed by a classification model and clustering model to determine efficacy. Finally, a hybrid model is created by combining a convolutional AE with a LSTM model. This supervised learning approach leverages the power of automatic feature extraction from the AE with the powerful temporal classification capabilities of the LSTM to reduce and extract temporal knowledge at the same time. The extracted features are further evaluated by a clustering algorithm. All implementations of the AE models initially used a literature inspired hyper-parameter setup. The hyper-parameters values were changed depending on experimental results.

4.1.1 Implementation of DEC

We trained the DEC AE model with partial data for one daily living scenario from the MobiAct dataset in Chapter 3. Before trying other AE models, we wanted to see if the DEC model performance improves when trained with the full dataset. We use the same architecture of the DEC model in this experiment while increasing the training epoches from 100 to 300, since the model must process more data. To compare and understand the goodness of the DEC algorithm, we also implemented it on the MNIST dataset to emphasize the capabilities of this approach and illustrate the greater complexities present in HAR datasets. The results for both HAR and the MNIST datasets are discussed in the results section below.

4.1.2 Implementation of 2D CNN + Autoencoder

To find a more powerful feature extraction model, we decided to use the classic Convolution Neural Network for exploration and ingestion on the HAR data. The idea

is to train the powerful CNN model using supervised learning and obtain classification performance, then if classification accuracy is good, the resulting features extracted from the CNN can either be used directly into the clustering algorithm or passed through an autoencoder for dimensional reduction before being used for clustering. In our experiments, we implement both methods and compared the outcomes. We first train the CNN model using supervised learning to perform classification and learn the key features from the data, then we combine the trained CNN model with an independent K-means instantiation for clustering. We evaluate the performance of clustering with and without AE dimensional reduction. The core idea is that belief that if classification accuracy is good, then the subsequent extract features should contain the core information required to distinguish activities.

The main challenge with this method is deciding how to feed 1D sequences into a model that only takes 2D arrays. Clever manipulation of the data is required to make each 'image' of HAR data 1) make sense architecturally, and 2) be usable by the neural network. Based on the work of [28], we decided to use 3 channels for the image arrays. Typically, the triple channels represent the Red, Green, and Blue hues in image data, but for our uses we will use them to represent the X, Y, and Z axis of the HAR data. Experimentation performed by both ourselves and Chen et al [28] determined this tensor structure to be the most ideal setup. Other variations result in a drop in accuracy up to 3%. We use a 1.5 second window, which translate to a total of 2,700 total points from all sensors. We split the the data points between the 3 channels and return 900 points per channel, meaning our array shape is an even 30 x 30. But because we split the channels by the 3 axis, we have to evenly distribute the data from the 3 sensors into each array. The final design is three (10 x 30) arrays

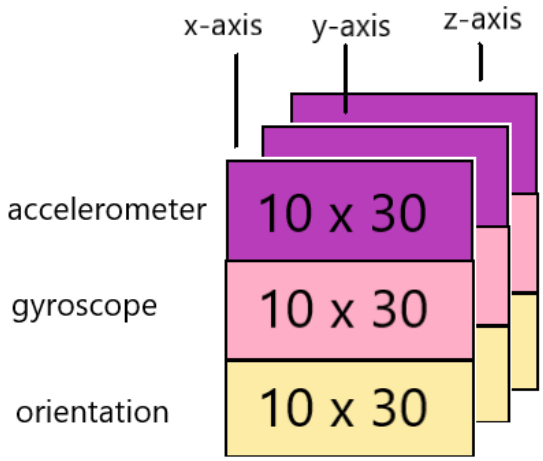

Figure 4.1: 2D Tensor for the MobiAct HAR data

stacked on top of each other, with each of the 3 arrays containing values from the accelerometer, gyroscope, and orientation sensor respectively (Fig. 4.1).

The 2D CNN is compiled with an Adam optimizer with a learning rate of 0.001 and the categorical cross-entropy loss function. The model consists of 2 stacked convolutional sections, a flatten layer, one dense layer, and a final classification layer. Each of the convolutional layers in the sections use the ReLu activation function and is followed by a maxpooling layer. Additionally, a batch normalization layer follows the maxpooling layer to add regularization. After the 2 convolutional sections, the model has a flatten layer to reshape the 2D tensors into a 1D array. This is fed into a dense layer, into a dropout layer, and into the output classification layer. The first convolutional layer has 32 filters and a kernel size of 3. The second convolutional layer has 64 filters and the same kernel size. Both maxpooling layers have a pool size of (2, 2) and stride of size 2. Prior to training, the data is normalized and split into training and testing data using a 80/20 ratio. Training the CNN took 100 epochs using a batch size of 128.

The AE architecture is very simple: it is a classic MLP (multi-layer perceptron) AE with a single encoding layer, a single latent representational layer, and a single decoder layer alongside the output layer. The encoding and decoding layers has 128 nodes and the model reduces the features into a representational vector of size 32. Every layer aside uses the ReLu activation function aside from the output layer, which uses a sigmoid function. The AE is compiled with the Adam optimizer and uses MSE loss. We trained the Autoencoder for 300 epochs. The higher number of training epochs for the AE is justified from our experiments showing that a much longer training time is required for the AE model to stabilize.

4.1.3 Implementation of LSTM Autoencoder

Next we wanted to incorporate some method of temporal information extraction from the AE. The previous 2D CNN-AE did not take into account temporal features, so perhaps that explains the poor clustering performance observed. In addition, the forceful restructuring of 1D time sequences in time-devoid 2D image tensors could have an affect on the resulting features, since the natural flow from sequence to sequences is disrupted. A LSTM-based autoencoder would both reduce the features and analyse temporal information at the same time. The datasets we use for this model are the full sequence MobiAct data, the segmented MobiAct data, and the UCI dataset. As a reminder, the full sequence data is using the generator approach, where the ultimate dataset is comprised of 15 smaller datasets, each corresponding to a single activity. The segmented MobiAct data is the raw dataset but reshaped into windows of size 200.

The architecture of our initial model is made simply of stacked LSTM layers, not

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 30, 30, 32)        2432
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 32)        128
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 15, 15, 64)        51264
_________________________________________________________________
batch_normalization_1 (Batch (None, 15, 15, 64)        256
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 512)               2097664
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 16)                8208
=================================================================
Total params: 2,159,952
Trainable params: 2,159,760
Non-trainable params: 192
_________________________________________________________________
```

Figure 4.2: Architecture of the 2D CNN

unlike the previously mentioned MLP AE we used in the 2D CNN experiments. The encoder portion of the model is made up of a LSTM layer with 128 nodes and the latent encoding LSTM layer with 64 nodes. The encoder is followed by a custom layer that reshapes the latent encoding into sequences capable of being accepted by the decoder. This is required because we are feeding in time sequences of variable lengths, so further processing is necessary to accommodate this fact. The decoder portion is the inverse of the encoder model, containing another LSTM layer with 128

nodes and the output dense layer. Every layer uses the 'ReLu' activation function aside from the output layer which uses the 'sigmoid' function. The simple LSTM AE uses the 'mse' loss function and the Adam optimizer.

Experiments with synthetic time-series data highlights the large amount of time needed for an LSTM autoencoder to learn the latent knowledge. Using this knowledge, we trained the LSTM AE for 300 epochs. The number of epochs was determined from similar literature models and our own experimentation. The reduced features are then passed through a MLP neural network for further training and classification.

4.1.4 Convolutional AE + LSTM

From our experiments using the CNN + AE framework and the LSTM AE (see section 4.2), the general conclusion seems to be that the AEs do not perform as well as pure supervised learning models. Intuitively, this makes sense as supervised learning has labels to guide the training process while the AE does not. Therefore, in the next approach we combine an AE trained using unsupervised learning with a LSTM classifier trained using supervised learning to create a hybrid feature extractor model. The first part of the model comprises a convolutional AE, which is similar to the previously implemented AE but uses the convolutional layers found in CNNs. This part of the architecture serves to automatically reduce and extract features from the dataset without any manual interference. The next part of the hybrid model features LSTM layers for temporal processing and multiple Dense layers for high dimensional representations and subsequent classification. The Conv AE extracts reduced feature set and the sequence of patterns returned by the Conv AE is learned by the LSTM for the classification of the activities. To clarify, this is a hybrid classification model

that uses supervised learning to train. This is different from the above implementations where we first trained an AE using unsupervised learning then transferred the features into separate analytical models. This implementation is a classifier model that first uses a convolutional AE in the first half of the architecture to reduce data complexity then feeds those reduced-activations into the LSTM classification layers. For this architecture, we use the raw dataset without manual feature engineering since the model should be capable of automatic feature extraction and we use the 'adam' optimizer alongside the 'categorical cross-entropy' loss function. The model architecture is shown in Figure 4.3. The trained classifier model is used as the feature extractor with K-means clustering and unsupervised learning to generate the clusters.

4.2 Results and Discussions

This section will present the results for all the AE models and provide a discussion for each implementation.

4.2.1 DEC

On the HAR datasets, DEC returns clusters scores of 0.54 NMI and 0.49 ARI for the UCI HAR dataset and 0.32 NMI and 0.24 ARI for the MobiAct data. From prior literature and our own experiments, we know that the DEC method is viable and powerful, achieving 84% accuracy and 0.84 NMI on the MNIST dataset [127] in the original paper and 91% accuracy and 0.91 NMI in our iteration. This result is also supported by the literature from which this DEC implementation came from [128] While we have previously determined that the MNIST is a simple dataset [11], preliminary results prove DEC can extract features despite the relatively simple design.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 128, 9)]	0
reshape (Reshape)	(None, 4, 32, 9)	0
time_distributed (TimeDistri	(None, 4, 15, 64)	1792
time_distributed_1 (TimeDist	(None, 4, 7, 64)	0
time_distributed_2 (TimeDist	(None, 4, 15, 64)	12352
time_distributed_3 (TimeDist	(None, 4, 960)	0
bidirectional (Bidirectional	(None, 200)	848800
dense (Dense)	(None, 300)	60300
dropout (Dropout)	(None, 300)	0
dense_1 (Dense)	(None, 100)	30100
dense_2 (Dense)	(None, 6)	606

Figure 4.3: ConvAE + LSTM Architecture

One reason for the disparity in performance could be that the classes (hand-written digits) in the MNIST dataset are inherently separable. From intuition and visualization of the MNIST data (something that we also demonstrate in Chapter 6), we can observe the clear borders between the digits. Models that excel in classifying MNIST digits are learning to isolate and learn the unique features of each class, but their job is made easier since each class is already quite distinct. On the other hand,

HAR classes (activities) naturally flows into each other. So, the algorithms tasked at classifying (and perhaps clustering) the data must metaphorically untangle the overlapping values into separate classes. In this case, adding even more data causes performance to drop even more. This is made more difficult when the data is presented in a pure 'value' perspective. As evident in Table 4.1, the results for DEC supports our assumption that perhaps temporal knowledge is required to effectively process time-series data.

Table 4.1: DEC Clustering Results.

Dataset	Clustering	NMI	ARI
UCI HAR	DEC (K-means)	0.54	0.49
MobiAct	DEC (K-means)	0.32	0.24
MNIST	DEC (K-means)	0.91	0.91

4.2.2 2D CNN + AE

The 2D CNN model achieves 98-99% accuracy provided that prior to training we upsample the lesser classes in the entire dataset using SMOTE (Synthetic Minority Over-sampling TEchnique) [27]. Without any sampling techniques, the classification accuracy drops to around 92-93%. Further experiments with upsampling just the training data and leaving the class distribution in the testing data untouched grants 94-95% accuracy. We believe that the third method is a more accurate representation of the real-life data. The upsampling is a good technique to account for the low number of observations for some activities - particularly the short, transitional actions such as CSI (Car-Sit-In) and falling actions. These movements takes place over a small window of time, so upsampling these data points allows the model to have access to

more knowledge. However, the frequency in the sampled data does not reflect the distribution in real data streams for HAR - no matter the scenario, more standard action such as walking or standing will outnumber the short transitional movements or any falling actions. Table 4.2 represents the 2D CNN classification results.

Table 4.2: 2D CNN classification.

Processing	**Classification**
Upsample dataset	99%
Upsample training	95%
No Upsample	93%

The K-means clustering performance when using a 2D CNN does not reflect the model's capabilities in classification. Although we are able to obtain high supervised learning accuracy, the resulting features and the corresponding clusters do not reflect this achievement. In reference to Table 4.3, clustering with K-means on just the CNN activations results in a 0.594 NMI and 0.441 ARI. Passing the reduced activations after using the autoencoder results in a slight drop in performance: 0.555 NMI and 0.340 ARI. These scores are similar to clustering with the statistical cluster algorithms. Notably, the autoencoder-reduced features observe a decrease in performance.

We surmise the reason for high classification versus low clustering performance has to do with the forceful reshaping of 1D time-dependent sequences into 2D tensors without time relations. Just from removing the temporal aspect, we inadvertently stripped an implicit set of features from the data. Our next steps are to account for the temporal relationships by implementing an LSTM AE.

As for the lowered performance after reducing the features through the AE, we believe the CNN has already extracted a greatly reduced representation of the HAR

data. Running these sets of features through the AE forced the model to further shave off information that is ultimately necessary. An analogy is trying to condense a material that is already compacted - there is a limit to how much we can further reduce the size, and at a certain point, we might be causing damage. We have also thought that the AE architecture is too simple, thus later experiments include more complex architectures or models that account for temporal information.

Table 4.3: 2D CNN Clustering results using K-means.

Data State	NMI	ARI
No AE reduction	0.594	0.441
AE reduced	0.555	0.340

4.2.3 LSTM AE

The resulting accuracy for the LSTM AE model is 72% after 150 epochs (out of 300). Additional epochs did not improve the accuracy score. Clearly, the performance is subpar compared to the previous methods. If classification accuracy is poor, then the chances that clustering performance on the same features being good are very low. Therefore, we have an incentive to maximize classification accuracy prior to clustering. The first change made to the architecture is replacing the LSTM layers with BiDirectional LSTM layers for increased computational power. This alone improved accuracy by 8 percent. As a second improvement, we used a LSTM classifier instead of a simple MLP. If the features extracted and reduced by the LSTM AE are temporal-based, then having a temporal-enabled model analyze and classify the sequences should result in better performance than the plain MLP classifier.

Our results seems to prove this to be true as accuracy reaches 89% for the MobiAct

dataset. Further experimentation with model architecture does not improve beyond this value, so we conclude that this is the best performance we can get with this method. It is worth noting that applying this model onto the segmented MobiAct data leads to an Out Of Memory error, which ends up crashing the process. As a result, no actual scores is given for those experiments.

Clustering using these temporal features with K-means grant us NMI and ARI scores of 0.489 and 0.310, respectively, and 0.598 NMI and 0.344 ARI with FISHDBC.

Table 4.4: Classification accuracy for LSTM AE.

Dataset	**Classifier**	**Accuracy**
MobiAct (Full)	MLP	72%
	LSTM	89 %
MobiAct (Seg.)	MLP	N/A
	LSTM	N/A
UCI	MLP	95%
	LSTM	93.2%

Table 4.5: Clustering scores for LSTM AE.

Dataset	**Clustering**	**NMI**	**ARI**
MobiAct (Full)	K-means	0.489	0.310
	FISHDBC	0.598	0.344
MobiAct (Seg.)	K-means	N/A	N/A
	FISHDBC	N/A	N/A
UCI	K-means	0.512	0.379
	FISHDBC	0.613	0.461

4.2.4 Conv AE + LSTM

With the MobiAct dataset, our initial test using the hybrid model returns 93.5 - 94% classification accuracy; this number is improved to a definitive 95% when we

introduced a 50% overlap in the data sequences, but we believe the underlying distribution between activities remains the same, and the increased performance is simply because the larger classes have become even larger. The experiments using the UCI HAR dataset provides similar accuracy of around 94-95%, but this is achieved only after substituting BiDirectional LSTM layers for increased computational power. It is worth noting that we could not recreate the performance (98% accuracy) reported by a similar model [116].

The features extracted from the UCI data enables near perfect clustering metrics (0.99 NMI and ARI) with K-means, showing that the method is capable of extracting valid temporal features and translating them into easily distinguishable clusters to a certain extent. For the MobiAct data, the model with K-means achieves clustering scores of 0.856 NMI and 0.719 ARI. To provide a quick explanation of the differing results between the UCI and MobiAct data, the reason the UCI data performs better is because the UCI data is much less complex than the MobiAct collection. The UCI HAR only has 6 activities, and the specific activities are the 'standard' expected ones such as standing, walking, and sitting. Additionally, the UCI dataset only comprises of around 10 thousand sequences, which is substantially less than the 80 thousand sequences present in our MobiAct implementation. As such, it is not surprising the clustering performance for UCI is far superior to that of MobiAct. Overall, the architecture performs the best out of all the AE implementations, which intuitively makes sense since it supplements training with supervised learning. The performance of the Conv AE + LSTM approach is highlighted in Table 4.6.

Table 4.6: ConvAE + LSTM results.

Dataset	Classification	Clustering Method	NMI	ARI
MobiAct	94%	K-means	0.856	0.719
		FISHDBC	0.743 - 0.808	0.678 - 0.832
UCI	95%	K-means	0.99	0.99
		FISHDBC	0.870 - 0.912	0.731 - 0.796

4.3 Comparing Clustering Performance for Reduced Activity

One question that arose from the experiments is why the UCI HAR dataset consistently outperforms the MobiAct dataset in almost every metric. An obvious answer is the simplicity of the UCI data which only contains 6 activities compared to the 16 activities in the MobiAct dataset. There is less overlap of values for different activities in the UCI data and the models can learn the patterns better. Another reason may be the lower collection frequency of the UCI data. To determine if fewer classes will improve clustering performance, we tested the hybrid model on a version of the MobiAct data that contains fewer activities. The idea is that certain actions contain inherent volatility just on the basis of what those actions are, and removing them can alleviate some uncertainty displayed by the model. The dataset we apply is the entire MobiAct data without any of the 4 fall activities, removing around 3% (around 1500 points) from the dataset - meaning the training and testing data only contains the 'core' 12 activities. We only choose the hybrid model for this dataset we feel any improvements should be immediately obvious in both classification and clustering scores.

4.3.1 Results

With the Conv AE + LSTM hybrid architecture, we achieved a classification accuracy of 95%, and 0.897 NMI and 0.831 ARI for K-means clustering. This is an improvement on the 0.806 NMI and 0.523 ARI when the full dataset was used. The clustering results using FISHDBC is 0.88 NMI and 0.86 ARI, which shows that the algorithm returns more consistent clusters compared to the previous scores of 0.87-0.91 NMI and 0.73-0.79 ARI. We see that while classification accuracy did not improve, the clustering accuracy clearly did, insinuating that although the fall activities are very minor in the grand scope of the dataset (3%), their impact on the interpretability of the cluster is quite substantial, showing that these minority actions does conflict greatly with the other activities. Since no other aspect of the pipeline is changed aside from the number of classes (and subsequently, the amount of sequences), the removal of the fall activities is what affected the clustering scores. Additionally, the increase in performance is not small enough to attribute towards the stochastic nature of the clustering algorithms. What is happening most likely is that since the fall activities are relatively minor and short in length, some of them were confused for activities with more presence, such as Walking or Standing. Tthe inclusion of fall data possibly confuses the model into learning the sub-par concept that fall actions might include other mixtures of similar activities. That would lead to the decrease in clustering performance. However, since the activities are not significant in the dataset, the classification score is not affected as much.

4.4 Comparative Performance of Clustering Models

We experimented the AEs with different clustering models. Clustering the HAR data with FISHDBC proves to be more challenging given that multiple parameters are required for the algorithm. The results are also more difficult to interpret given the ability of FISHDBC to classify points as noise since the algorithm is based off DBSCAN. While normally this ability can be useful, in this situation it is not ideal, since none of the points are originally labelled as noise. Therefore, the scenario we find ourselves in is either: 1) the algorithm returns the appropriate matching number of clusters at the cost of a large number of points classified as noise in addition to potentially lowered performance, or 2) we return higher performance in exchange for an inaccurate number of clusters and typically reduced noise. We also observe that the lower frequency movements such as any of the fall actions end up being encapsulated by the higher frequency movements such as walking and/or standing. Since the less frequent actions represent an insignificant portion of the data, even if they are mistaken for a more frequent action the clustering score is not affected as much.

4.4.1 Results

For the MobiAct data, scenario 1 (appropriate number of clusters with high noise) provides cluster scores of 0.743 NMI and 0.678 ARI, and one instance of scenario 2 (high performance but inaccurate number of clusters) returns 0.808 NMI and 0.832 ARI with only 6 clusters out of 16 returned. Different parameters fed into FISHDBC returns different instances of scenario 2 at varying number of clusters and performance metrics. With the UCI data a similar trend is observed. Scenario 1 gives the right

number of clusters but at the expense of lower scores of 0.870 NMI and 0.731 ARI. Scenario 2 provides much better performance of 0.912 NMI and 0.796 ARI with almost no noise but drops a class.

4.5 Further Discussion

Several observations are evident from the experimental results derived with the AE models on the HAR datasets. First, even with access to more powerful architectures, more training observations, and temporal information, the AE models are incapable of effectively returning distinct features for clustering. We believe there is a compressional limit to the feature reduction that the AE models can perform on the data streams, or perhaps fundamentally unsupervised learning neural networks cannot extract good clustering features from timestamped HAR IoT data. Second, the model that succeeds in good clustering performance is the Conv AE + LSTM hybrid model. This is determined by the high clustering scores compared to the other two AE models and the standalone clustering models. The resulting clusters returned by the Conv AE + LSTM architecture outperforms all previous models by at least 0.30 points for both NMI and ARI, with the most obvious and notable improvements on the UCI dataset. The hybrid model obtained near perfect clustering scores with the UCI data using K-means and saw a rise in performance with the density-based FISHDBC algorithm. This architecture is unique because it implements a supervised learning approach in combination with temporal feature process from the LSTM layers. From these observations and based on the results from the standalone clustering experiments, we believe that deeper architectures are not the sole necessity for effective clustering of HAR sequences; powerful supervised learning architectures in addition

to some aspect of temporal feature processing is required to provide good clustering results for HAR.

4.5.1 Insights on Cluster Hierarchy

We believe that the density-based clusters grant us insight into the hierarchical nature of the activities. By observing which clusters are absorbed into the dominating clusters and in which sequence, we can determine the similarities between activities based on the order of assimilation. For example, if we take the second scenario (high performance but inaccurate number clusters) for the UCI data and map the difference in cluster assignment for each of the points from scenario 1 (appropriate number of clusters with high noise), we can gain insights about which sequences of observations are the most similar in terms of temporal features. Let us propose that in scenario 2, cluster number 5 disappears and all its contents are re-assigned to other clusters. Perhaps in this case, the elements of cluster 5 are absorbed entirely by cluster 4. If we can map out which cluster belongs to which activity, then we gain knowledge that the observations from that activity are closest to the action represented by cluster 4 at a deeper level.

Similar insight can be extracted from the MobiAct data, albeit the task is more difficult given the number of activities and the inherent complexity in the data just from the volume alone. The parameters for the clustering algorithm can be manipulated such that the number of clusters steadily decreases at each iteration, at which point a mapping function is deployed to note the cluster assignment differences compared to the previous iteration. A hierarchical representation of the clusters can be extracted with this method and further analyzed for innate knowledge not evident

from classification or surface level clustering, such as hidden clusters bridging between two activities or unexpected associations between other actions. However, the exact details of this approach is beyond the scope of this work.

4.6 Summary

From the various experiments presented in this chapter, we infer that the AE models are not able to effectively reduce the features in a way that creates clear and distinct concepts to use for clustering. We implemented multiple AE architectures, used classification with supervised learning to train different feature extractor models and then used the trained model with unsupervised clustering, and experimented with multiple data sets having different number of activities. The results demonstrate that a hybrid model with a Conv AE and LSTM classifier architecture trained using supervised learning serve as a good feature extractor for a K-means clustering model. Since the LSTM model learns temporal information, from our experiments with the 2D CNN and DEC, we believe that not only do we require a powerful model architecture, but temporal features in combination with supervised learning must be utilized in some capacity to effectively cluster HAR data. Therefore, next we explore temporal feature extraction techniques further to improve the clustering results.

Chapter 5

Temporal Feature Extraction for Clustering HAR Data

Based on the results of the experimental study presented in Chapter 3, we decided to explore the temporal pattern learning techniques further to improve the clustering quality. Previous chapters demonstrate the computational limits of purely statistical methods and even highlights the weakness of powerful architectures such as 2D Convolutional Neural Networks and Autoencoders. Without any temporal context, the previous algorithms must navigate through a high dimensional feature space where seemingly every point is similar to every other point. Additionally, in a timeless environment, important features such as change in speed or consistency of velocity is ignored. Gradual changes in sensor values are too insignificant in the context of clustering HAR data. The temporal knowledge is also insufficient as demonstrated by the LSTM AE in the previous chapter. Therefore, the lackluster performance of prior models in combination with the high achievements of the hybrid AE model implicates the dire requirement for temporal feature extraction with some form of supervised learning. In this chapter, we explore the concept of Temporal Feature

Extraction (TE) [77, 22], a technique to formulate core temporal features from time-series sequences. We present 4 evolutions of the TE model architecture and perform exploratory experiments to determine their effectiveness on HAR time series data using two metrics: 1) the classification accuracy and 2) the clustering performance on the extracted temporal features.

The following sections will be as follows: The temporal extraction models are described in section 5.2, but we also include the classification and clustering results and pertinent discussions relating to that specific model in that model's section. Section 5.5 covers general discussions applicable across all the models and we conclude this chapter with section 5.6.

5.1 Datasets

The two datasets we use in this Chapter are the full MobiAct and the UCI HAR datasets. However, due to architecture limitations, we will have various iterations of the MobiAct dataset. In the section below, we will explain the various preprocessing methods we apply to the MobiAct data including the structure and format of each model that uses them. The UCI HAR dataset remains unchanged throughout this chapter.

5.2 Data Engineering for Temporal Feature Extraction

The preprocessing for time-series data is tedious and subject to trial-and-error largely because of the chaotic nature of time data and the imprecision of the corresponding equipment. For example, time segments can be of varying lengths, forcing researchers to develop methods to equalize unmatched sequences. The sensors used for recording

can sometimes register more values than expected, leading to further problems to address. Additionally, datasets originating from dissimilar sources adopt particular formats, which require some adjustments. Table 5.1 contains a quick summary of the methods we applied to the MobiAct dataset.

5.2.1 Data Size Reduction

In the interest of saving space and computation time, initial experiments read every 3rd activity file for each activity. Since each person performs the activity thrice, this means the dataset comprises of one examples from each participant for each activity. Later experiments use the entire dataset, but no significant performance improvement is observed while run-time greatly increased. For context, depending on the method of processing, the entire MobiAct dataset could be as large as 4GB in size - this can make some operations that require loading the dataset into memory difficult. The majority of the MobiAct iterations do use the entire dataset. The UCI dataset is much smaller in size, so no such action was necessary.

5.2.2 Data Transformation

Many-to-many, also known as sequence-to-sequence (seq2seq), and many-to-one are two of the most common transformations applied to time-sequence data when used with LSTM models. LSTM networks applies the above transformation at each stage of iteration, and the shape is usually reflected in the final output layer. Seq2seq means that if we feed in a sequence into the model, we receive back a full sequence. For our uses, a trivial example would be passing a sequence of shape (observation, time step, features) and we get another sequence of shape (observation, time step, label) as the

output. This means the model assigns a label to every timestep in the observation. Conversely, if the output is the observation or label, then the model assigns a label to the entire observation sequence, implementing the many-to-one architecture. To provide a numerical example, let us say the input sequence is (100, 20, 9), meaning this array has 100 observation each with 20 timesteps and 9 features. So, the model effectively reads in an array of shape (20, 9) at every iteration. In a seq2seq design, the model will output a sequence shaped as (100, 20, 1), assigning each data point a distinct label for a total of 2000 (100x20) data points. In actual implementation, the number representing the labels will not be 1, since for training purposes we will apply one-hot encoding to the classes. This means that if we had 3 classes in the dataset, we will represent class 0 as [1, 0, 0]; class 1 as [0, 1, 0]; and finally class 2 as [0, 0, 1]. Therefore, the true shape of the output sequence will be (100, 20, 3), for example. In a many-to-one implementation, the shape of the output sequence becomes (100, 3), because the model assigns a singular label to the entire sequence rather than to the individual points. In a many-to-one design, the model views the 20x9 array as a singular object, contrasted with the seq2seq design, which places a priority on the individual elements. The LSTM implementations we conduct in this chapter will use a mixture of seq2seq and many-to-one architectures.

5.2.3 Feature Engineering

One of the lessons derived from the past experiments with the statistical models is the demand for some application of feature engineering. Raw sensor values do not seem to contain enough information for algorithms to fully understand the underlying patterns within the dataset. This is especially relevant when an observation only

contains 9 features - all of which encodes an integral part of that observation. By engineering additional features from the existing ones, any data reduction/feature reduction algorithm now has a larger selection to work with.

For the MobiAct dataset, we decided on a window size of 1 second, and since the data was recorded at 200Hz, this meant aggregating every 200 observations to create a single engineered time-step. According to [100] and [109], the following features were shown to be ideal for HAR: mean, median, standard deviation, kurtosis, variance, skew, min, max, zero crossing rate, and slope. Originally, the plan was to extract one feature from all axis of a sensor (i.e. use the X, Y, Z accelerometer axis to extract the mean) and create a dedicated feature for that sensor. However, we noticed the variance for the 3 axes varies quite substantially. For example, one sample of accelerometer data is [-0.49, 9.8, -0.47]. Creating a mean feature from these value will not be an ideal representation. Our approach is to apply feature engineering on each of the sensor axis, with some features made from a combination of other features. For every window slice, we take the values of a single axis of a sensor and aggregate them into a single feature. For example, the first manually engineered feature will be the mean of the X axis values for the accelerometer in the current 200-length observation window. The next engineered feature is the median of the set of data points. We repeat this process for all 9 feature columns and create some additional features based on several newly created features. Thus the final shape of a single observation becomes (1, 82), from the original shape of (200, 9). The label for each aggregation will be the activity with the highest frequency. Depending on our implementation of this method, we can choose to create a dataset for a many-to-one architecture or a seq2seq architecture. The final data shape in the many-to-one

becomes (n, m, 82) and the label shape follows with (n, 16). If constructed for a seq2seq model, the data shape stays the same but the label changes to (n, m, 16)

5.2.4 Data Aggregation

We previously mentioned that one of the problems with time-series data is that typically the data sequences are frustratingly imperfect - some sequences belonging to the same group can differ in length, or the time-steps between groups can be drastically different. This is problematic because 1) the data requires additional processing checks and 2) this prevents simply passing the data into the model. Ordinarily, the entire training data can be passed as input into the neural network, but clearly we are unable to do so as the various lengths of sequences prevents the creation of a universal array.

There are three main methods used to address sequences of differing lengths, each with its benefits and detriments:

- Appending trivial data to the beginning or end of the short sequences (Padding)
- Removing data from the beginning or end of the longer sequences (Truncation)
- Scaling shorter sequences to match the longer ones (Scaling)

Padding

If we apply padding to a sequence, we risk the model learning large segments of trivial data if the length disparity between observations is very large. For example, after we perform manual feature aggregation on the MobiAct data, we end up with BSC sequences of shape (1, 20, 82) and WAL sequences of shape (1, 300, 82). If we pad

the BSC sequences with trivial '0' data, we end up adding 280 segments of pure 0s to pad a BSC sequence to 300. This can have an effect on performance depending on the model architecture we choose.

Truncating

Truncation of data means removal of useful information. Take the WAL example above, if we want to truncate the sequence to fit the shape of BSC, we would be removing 280 segments of WAL data. Perhaps for WAL, the majority of it is redundant information, but this issue might come up for other activities with long sequences as they could be hybrid actions consisting of multiple activities in one sequence.

Scaling

Finally, scaling shorter sequences initially sounds plausible, but the problem arises when we encounter the quick activity transitions mentioned before. If normally, a transition takes place in half a second, scaling shorter sequences will greatly lengthen the transition segments, potentially destroying the latent information carried within the half-second.

The solution is to create individual datasets that corresponds to each activity, then we feed these sequentially rather than combining the actions together and needing to figure out a compromise between the three variable sequence lengths. Training models with the generator approach simply requires defining the number of steps (which corresponds to the number of activities) in an an epoch and setting the generator to loop through the activity datasets endlessly. The result is 15 datasets corresponding to the number of distinct activities, with each dataset containing only the sequences

for that activity. While the MobiAct data has 16 activities, LYI (Lying down) does not have its own distinct data, and its sequences are derived from existing activities. The complete dataset becomes 15 * (m, n/200, 82) where m is the number of files for each activity, n is the number of timesteps in each file, 82 are the manually constructed features, and 15 is the number of activities. Each of these datasets will have a different shape from other datasets, since their sequence lengths are not the same. We can choose to construct a seq2seq dataset or a many-to-one dataset depending on our needs.

5.2.5 Use Raw Data: No Data Preprocessing

A simpler method of data processing is to avoid manual feature engineering and present the data in its raw form. Note that 'raw' in most cases refers to a pre-processed version of the HAR sensor values given by the original researchers. The actual raw data taken directly from the live sensors are not usually given. However, since LSTM networks require sequences to be in a specific form, a small amount of preprocessing is still necessary even with the raw values. We still want to segment the observations into smaller sub-sequences so the 200 sequence window is re-used, with the difference being the lack of aggregation. Instead of combining an array of shape (200, 9) into (1, 82), we simply frame the entire 200 window as one observation, creating a shape of (1, 200, 9). For example, if one file of activity A has 1000 observations (1000, 9) and one file of activity B has 2000 observations (2000, 9), the resulting data arrays are shaped (5, 200, 9) for activity A and (10, 200, 9) for activity B, making the total data array a shape of (15, 200, 9). As shown, this method allows for the stacking of classes which the previous generator method could not do. If required, further processing can be

performed on this data to create manual features. Taking the previous example, we can change the (15 observation, 200 timestep, 9 feature) array and return a feature-engineering array of shape (15, 100, 82) by aggregating every two observations. Of course, aggregating 200 observations will result in (15, 1, 82) which is essentially the same as just manual engineering.

Note that the UCI dataset comes available in this form, so further data shape processing is not required to fit into the LSTM models.

Table 5.1: Processing methods on the MobiAct dataset.

Method	Original shape	New shape	Shape of dataset
Feature Engineering	(n, 9)	(n/200, 82)	(m*(n/200), 82)
Generator	(n/200, 82)	(m, n/200, 82)	15*(m, n/200, 82)
Raw Data	(n, 9)	(n/200, 200, 9)	(m*(n/200), 200, 9)

5.3 Material and Methods

5.3.1 Standard TE Model

The first TE model has a very basic architecture and correspondingly applies a basic data preprocessing. The purpose of this model is to establish a baseline for future comparisons and establish that the method has merit. Our base TE architecture consists of a single 1D Convolutional layer, a LSTM layer, and a Dense classification layer, all of which can be seen in Fig 5.1. All subsequent iterations of TE implementations will be built on this design. The hyper-parameters for this implementation took inspiration from similar literature models. Subsequent evolutions of the TE architecture inherit the hyper-parameters used here.

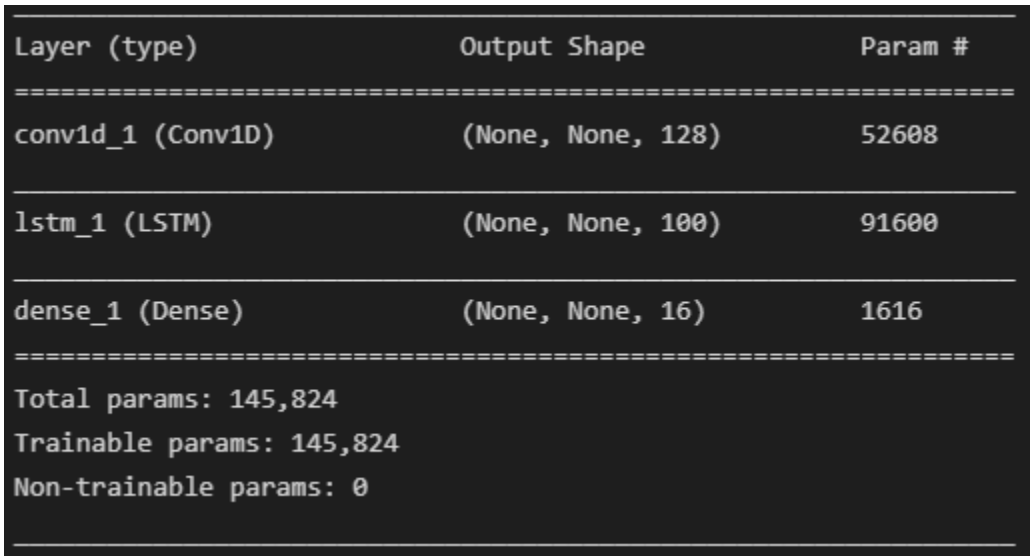

Figure 5.1: Base temporal extraction model

Implementation

For MobiAct data preprocessing, we address the problem of unequal lengths using a naive approach. Take two activities, BSC and WAL, as an example. If we read in the first file for each activity, after aggregation and feature engineering, the BSC file returns a shape of (20, 82). However, the WAL file returns a shape of (200, 82). One simple way to solve this is to append all the sequences together and create an array of shape (220, 82). We reshape the array into (220, 1, 82), since that is the required shape for the LSTM model.

For the purposes of testing concepts, we append all aggregated sequences together to feed into the neural network. Prior to training, we scaled the data with the MinMaxScaler to limit the range of values between 0 and 1. This method is picked because the features are scaled column-wise, and since each sensor has a difference min/max value, the columns are scaled based on their respective sensor. We train for 64 epochs using a training/testing split of 80/20 and a batch size of 64. The

model uses the categorical cross entropy loss function with the Adam optimizer and a learning rate of 0.001.

5.3.2 Improved TE / Time Slice Model

This model is a direct improvement on the base TE design. In addition to adopting a more complex architecture, the data preprocessing is further honed to improve model performance. Training and testing results show a marked increase in performance. Several iterations with varying parameters were tested - a couple notable instances are recorded below. The architecture saw the inclusion of an additional 1D Convolutional layer and fully connected Dense layer. The first part of the model uses the power of stacked convolutions with decreasing filters and kernel sizes to extract a latent representation of the sequence. A drop out layer is added to help with overfitting. This leads into two LSTM layers and finally into a Dense layer, another dropout layer, and the final classification layer. The improved architecture is shown in Fig 5.2.

A notable inclusion to the architecture are the two dropout layers located after the Convolution layers and the first fully connected Dense layer. We can see from the base model results that it severely overfits to the training data. We can tell by noting the difference in magnitude between the training accuracy and the validation accuracy. High training and validation performance means the model is capable of generalizing features to fit previously unseen data, while drastic differences between the two values (typically high training and lower validation) implies overfitting to the training data. The prior model outperforms on training data by around 10 percent. The dropout layers are added to combat this. Dropout works by randomly setting units (depending on a parameter) to be 0, essentially forcing the model to 'forget'

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, None, 128)	52608
conv1d_1 (Conv1D)	(None, None, 64)	24640
dropout (Dropout)	(None, None, 64)	0
masking (Masking)	(None, None, 64)	0
lstm (LSTM)	(None, None, 128)	98816
lstm_1 (LSTM)	(None, None, 64)	49408
dense (Dense)	(None, None, 100)	6500
dropout_1 (Dropout)	(None, None, 100)	0
dense_1 (Dense)	(None, None, 16)	1616

Total params: 233,588
Trainable params: 233,588
Non-trainable params: 0

Figure 5.2: One iteration of the improved TE architecture

certain features. This prevents any nodes from dominating during training and to help spread out data generalization across the entire network. The placement of the dropout layers do not matter too much so long that they are not directly after any of the LSTM layers. Recalling back, LSTM nodes pass on their states to the next iteration in the batch. They also take all values into account, meaning the random nodes affected by dropout (and set to 0) will be passed forward, where it will be

considered significant to that observation sequence. That is why the dropout layers are applied after the Convolution and Dense layers. The LSTM implementation in Keras/TF has their own version of dropout called 'Temporal Dropout' that is specially designed to remove the aforementioned problem.

Implementation

We trained the improved TE model using the same conditions as our base model. The only difference is in the parameters within each layer and the way we feed in training data. While the prior TE model simply used a stacked datasets devoid of deeper processing, this new architecture uses the generator approach explained above. To re-iterate, the generator approach returns an infinite loop of 15 datasets each pertaining to a single activity. This means during training the model is trained on each activity first before moving onto a different one. We will call this dataset the 'Full Sequence MobiAct'. The improved architecture 93% classification accuracy on this data. We also tested using the UCI HAR data, with the model returning around 92.5% accuracy.

We explore the model by changing parameters, adding layers, and applying various concepts to improve performance. The first major change is switching the regular LSTM layers into Bidirectional LSTM (BiLSTM) layers. At its core, LSTM networks preserve the information from inputs that have already passed through its nodes by storing them in the hidden states. This information is only preserved from the past, since that is the only input that the network has seen. The benefit BiLSTMs provides is the capability to preserve information from both the past and the future by running inputs forward and backwards. It contains an additional hidden state it uses to store

future features. The added capability allows BiLSTM networks to extract deeper context. In this scenario, the model is capable of looking ahead to see what values come after before making a decision for classification.

5.3.3 TimeDistributed TE Model

This architecture is not drastically different from the other TE designs aside from allowing non-LSTM layers access to temporal features through the use of Keras/TF's TimeDistributed layer wrapper. What this does is grant layers wrapped by TimeDistributed to process an additional dimension for its input data - time. In our case, we apply it onto the Conv1D, dropout, and flatten layers. To allow this architecture to work, we also have to add the corresponding time dimension onto our dataset. Let us say our original data shape is (100, 20, 10). If we want this data to be applicable for a TimeDistributed TE model, we must reshape the data into the form (100, t, 20/t, 10), where t is the chosen number of 'time slices'. Say we pick t=4, the resulting data becomes (100, 4, 5, 10). When a TimeDistributed wrapped layer receives a batch of data, the model will apply the same instance of that layer onto each of the 4 time slices independently; the same set of weights are used for each of the timesteps. Although the 1D CNN is typically used for time-sequences, it does actually account for temporal information. Wrapping it in a TimeDistributed layer allows the 1D CNN to create time-dependencies across samples, therefore potentially discovering and extracting more temporal knowledge. The TimeDistributed architecture is presented in Figure 5.3.

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 4, 124, 128)	5888
time_distributed_1 (TimeDist	(None, 4, 122, 64)	24640
time_distributed_2 (TimeDist	(None, 4, 122, 64)	0
time_distributed_3 (TimeDist	(None, 4, 61, 64)	0
time_distributed_4 (TimeDist	(None, 4, 3904)	0
bidirectional (Bidirectional	(None, 4, 128)	2032128
bidirectional_1 (Bidirection	(None, 64)	41216
dense (Dense)	(None, 128)	8320
batch_normalization (BatchNo	(None, 128)	512
dense_1 (Dense)	(None, 16)	2064

Figure 5.3: TimeDistributed TE architecture

Implementation

For our MobiAct experiments, we use decreasing filter sizes for the TimeDistributed 1D CNN (128, 64) and correspondingly decreasing kernel sizes (5, 3). Both of the 1D CNN layers share the 'ReLU' activation function. These two layers are followed by a dropout layer with 0.5 dropout (meaning 50% of nodes are forgotten) for regularization. A 1D maxpooling with poolsize of 2 and a flatten layer then process

the data and reshape it for LSTM input. From various test, we discover that BiDirectional LSTM (BiLSTM) layers grant a small increase in performance, so for the TimeDistributed models, we decide to exclusively use BiLSTMs over the regular implementation. The number of BiLSTM nodes ranges between 32 and 64, but most of our experiments use 32 alongside a recurrent dropout of 0.5. Recurrent dropout is the similar to regular dropout but designed specially for temporal layers. A dense layer with 128 nodes and a 'ReLU' activation function receives the features extracted by the BiLSTM and creates a high level representation before passing the knowledge through a batch normalization layer and into the final 'Sigmoid' classification layer. The model uses the Adam optimizer and categorical crossentropy loss. For all our tests, we use an epoch and batch size of 64.

Results and Discussion

5.3.4 Multi-Headed Temporal Extraction

This iteration of TE is similar to the previous implementation but with three CNN heads for added feature exploration power. The motivation is to grant the model various levels of granularity by varying the kernel size in the CNN heads. As a result, the model gains three difference perspectives on the same sequences, and should theoretically extract greater amounts of knowledge from the data. The downside to this design is the increased computational resources taken up by the model to balance the cost of the extra convolutions. The three CNN heads each adopt kernel sizes of [5, 7, 11] respectively and is followed by a typical Dropout, MaxPooling, and Flatten layer to regularize, reduce, and reshape the sequences for the following LSTM layers. In the main implementation, we use two BiLSTM layers for increased ability.

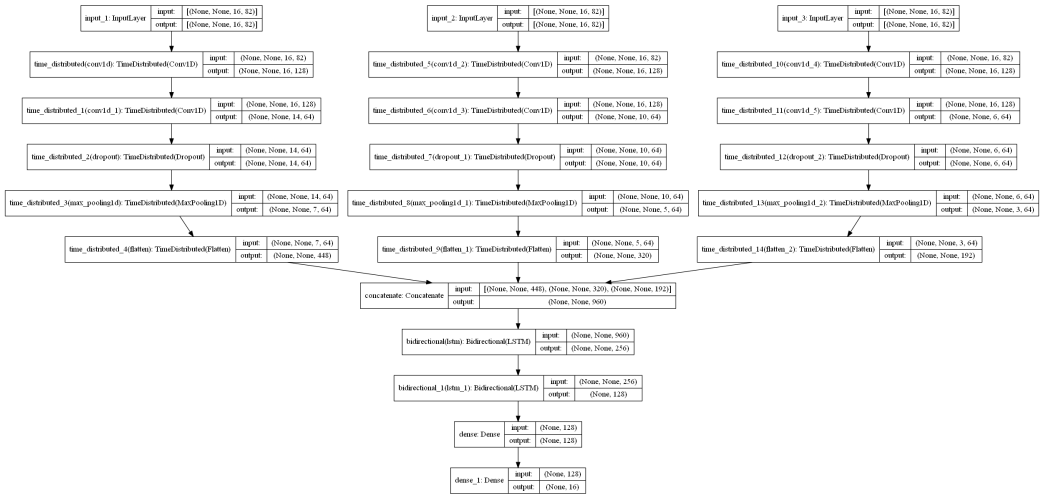

Figure 5.4: Multi-headed CNN into BiDirectional LSTM

A Dense layer is included after to create a high dimensional representation of the observation. After, the data is fed through a batch normalizatoin layer then into the final classification Dense layer. Recurrent Dropout is used in the BiLSTM layers to combat overfitting, a technique that is necessary given the tendency for this powerful model to overfit on the training data. The architecture is shown in Figure 5.4.

Implementation

Since the Triple-Headed model is similar to the TimeDistributed TE model, the prior datasets can be recycled for this new iteration. One caveat is that since we have a triple head for the CNN portion, we must feed in an array of datasets rather than a single dataset. As a result, the input is: [dataset, dataset, dataset], where each index is just an instance of same dataset. The classification accuracy is provided in Table 5.7 and the clustering performance is given by Table 5.8.

5.4 Results and Discussion

This section presents the results for all the TE models.

5.4.1 Standard TE model

The model achieves a training accuracy of 94% and testing score of 92.6% when fed data with the many-to-one form. Subsequent experiments with the same model on seq2seq data returns the same performance, implying that perhaps either methods of data structure are just as valid. We also applied the same model onto the UCI data, which only comes in a many-to-one form. After normalizing with StandardScaler, the simple TE implementation returns a 96.1% training accuracy and a 92.4% testing accuracy on the UCI data. Table 5.2 lists the classification accuracy for the model. From these results, we can confirm the method's validity for HAR data and proceed with designing more complex design to attain higher performances.

Table 5.2: Base TE classification accuracy.

Dataset	Classification
MobiAct (many-to-one)	92.6%
MobiAct (seq2seq)	92.7%
UCI	92.4%

5.4.2 Improved TE model

The BiLSTM model achieves a testing accuracy of 94%. According to [100], overlapping windows can have an improvement model performance. The Bidirectional model is fitted with 50% overlapping data and it returned a 1 percent point improvement at 95%. Table 5.3 presents the model hyper-parameters and architectures we explored.

Notably, increasing the filter size in the 1D CNNs and the number of nodes in the BiLSTM layers did not produce a change in performance.

Table 5.3: Performance results for the improved TE model.

Model Parameters	Training Acc	Validation Acc
Base	98%	93%
BiLSTM	97%	94%
BiLSTM + overlap	97%	95%
↑ filters + ↑ BiLSTM nodes	97%	94%

Before we continue, we should note the reason other versions of the MobiAct data is not mentioned for this model. With the Full Sequence MobiAct, the model takes in an entire activity file and creates temporal features from that single sequence. A generalization is created for the full length sequence and the model achieves this by effectively summarization the information within the data. On the other hand, with the prior MobiAct datasets, the model must generalize across multiple sub-sequence and create multiple summaries for many more piece of data. This means that the amount of processing required is much higher. When we tested this architecture using a dataset other than the generator method, we noted that the time taken to process each sequence could range from 2 minutes to 5 minutes each, depending on how we processed the data - this is very expensive for processing a single sequence. As a result, we decided this architecture is not appropriate for certain types of datasets so we did not include the results for those. The purpose of this model is to set a baseline for the subsequent models to compare after training on the Full Sequence MobiAct.

5.4.3 TimeDistributed TE model

The results of the TimeDistributed TE model are listed in Table 5.4 and the corresponding clustering performance in Table 5.5. Several variations of the MobiAct data is made available using this model architecture and we try to apply as many as we can on this iteration because this version of TE is both powerful yet not overly complex, meaning we have a good chance to observe the best performances without encountering lengthy training times. In the order in which they appear in Table 5.4, the first dataset design we use is the many-to-one sequencing for the MobiAct dataset. The raw data is simply reshaped from (n, 200, 9) into the appropriate form (n, 10, 20, 9) and after scaling with a MinMaxScaler, is fed into the model for training. We get back a classification accuracy of 93.1% and clustering scores of 0.664 NMI/0.628 ARI using k-means and 0.564 NMI/0.372 ARI using FISHDBC. Next, 'Engineered MobiAct' is the the raw dataset after applying manual feature generation on the windows. For that dataset, we aggregate every second observation to transform (n, 200, 9) into (n, 100, 82). This method performs roughly the same in classification (93.6%) and notices a slight drop in ARI clustering performance. The 'Full MobiAct' is unique in a sense that instead of segmenting each activity file into smaller sequences defined by a window, we treat the entire file as a single sequence. This is also the dataset we employ the generator method for, since each activity sequence ends up being different lengths. This method of processing performs the best in both classification and clustering, returning near perfect scores for both. Finally, we applied the UCI HAR dataset, which is in the same form as the 'Raw MobiAct'. This dataset returns a 92.2% classification accuracy but a much higher clustering performance than the 'Raw MobiAct', with 0.927 NMI and 0.29 for K-means and a 0.860 NMI and 0.874

for ARI using FISHDBC.

Table 5.4: TimeDistributed model classification accuracy.

Dataset	Classification
Raw Reduced MobiAct	93.1%
Engineered Reduced MobiAct	93.6%
Full MobiAct	98%
UCI	92.2%

Table 5.5: TimeDistributed model clustering scores.

Dataset	Clustering Alg.	NMI	ARI
Raw Reduced MobiAct	K-means	0.664	0.628
	FISHDBC	0.564	0.372
Engineered Reduced MobiAct	K-means	0.649	0.431
	FISHDBC	0.543	0.348
Full MobiAct	K-means	0.99	0.99
	FISHDBC	0.98	0.98
UCI	Kmeans	0.927	0.929
	FISDHBC	0.860	0.874

As an additional note, Table 5.6 below shows the classification results on the full sequence MobiAct data in seq2seq form without any normalization. Multiple architectures are used in combination with different hyper-parameters. We can see that the performance stagnates at around 90%. This shows the importance of normalizing data - especially if the values come from different sources. In this case, we scaled each input vector individually to vector length.

5.4.4 Tri-Head model

We can see that the accuracy performance using the Tri-Head TE model only gains a slight improvement over the regular TimeDistributed TE model, going from 93.3% to

Table 5.6: Varying architecture results on unnormalized full MobiAct seq2seq data.

Architecture	**Result (Acc.)**
Double BiLSTM (128, 64)	91%
Double BiLSTM (64, 32)	89%
Single BiLSTM (64)	85%
Single BiLSTM (32)	90%
Single BiLSTM (16)	90%
Single BiLSTM (16) + Conv(64,32)	85%

Table 5.7: Classification accuracy with Tri-Head model.

Dataset	**Classification**
Raw MobiAct	93.8%
Engineered MobiAct	93.8%
Full MobiAct	98%
UCI	93.5%

Table 5.8: Clustering performances using Tri-Head model.

Dataset	**Clustering Alg.**	**NMI**	**ARI**
Raw MobiAct	K-means	0.749	0.522
Raw MobiAct	FISHDBC	0.634	0.648
Engineered MobiAct	K-means	0.784	0.548
Engineered MobiAct	FISHDBC	0.684	0.673
Full MobiAct	K-means	0.99	0.99
	FISHDBC	0.98	0.98
UCI	K-means	0.99	0.99
	FISDHBC	0.94	0.93

93.8% on the raw MobiAct dataset. Accordingly, the engineered data provides similar scores of 93.6% for the TimeDistributed model against the 93.8% of the Tri-Head architecture. For both the Tri-Head and TimeDistributed designs, the full sequence MobiAct returns high classification. Seemingly, both models are practically identical in terms of classification, which raises the question whether the more powerful and subsequently resource hungry Tri-Head implementation is necessary for HAR feature extraction when the simpler model performs just as well and with more efficiency. As shown in Table 5.9, the time taken to process a sequence using the Tri-Head model is more than twice the amount taken by the previous iteration at around 48 seconds.

Table 5.9: Time taken to process a batch of 128 sequences.

Architecture	**Time**
TimeDistributed	21 seconds
Tri-Head Model	48 seconds

However, the clustering performance between the two designs shows a different story, as Table 5.8 displays the clear increase in metric scores for the Tri-Head model across all the datasets. From this, we can conclude that the newest iteration of TE is capable to extracting more applicable knowledge for clustering algorithms at the cost of greater resource draw. Presumably, the use of three levels of analysis on each sequence returns features which are more easily distinguishable by the simple distance metrics the clustering algorithms use. We can notice this fact reflected in the slight performance increases in the Tri-Head classification accuracy scores - in the overall picture the additional knowledge might not affect weight changes in the TE model greatly, but if taken into the context of distance metrics, the change is quite significant.

As an additional note, we wish to bring up the training timeline of the Tri-Head model on the full sequence MobiAct data. As shown in Figure 5.5, this model in combination with this dataset iteration begins with very low accuracies before gradually achieving near perfect performance. This is contrasted with all the other dataset iterations, where the trend is high initial accuracy (75%) and a quick tapering in performance. We believe this dataset is unique because we are asking the model to learn and connect temporal dependencies across features from an entire activity file rather than independent sub-sequences. As a result, the first half of training takes longer since the full sequences can be drastically difference in values and lengths. Eventually, we do see the accuracy stabilize at 97-8% as the temporal dependencies are finally learned.

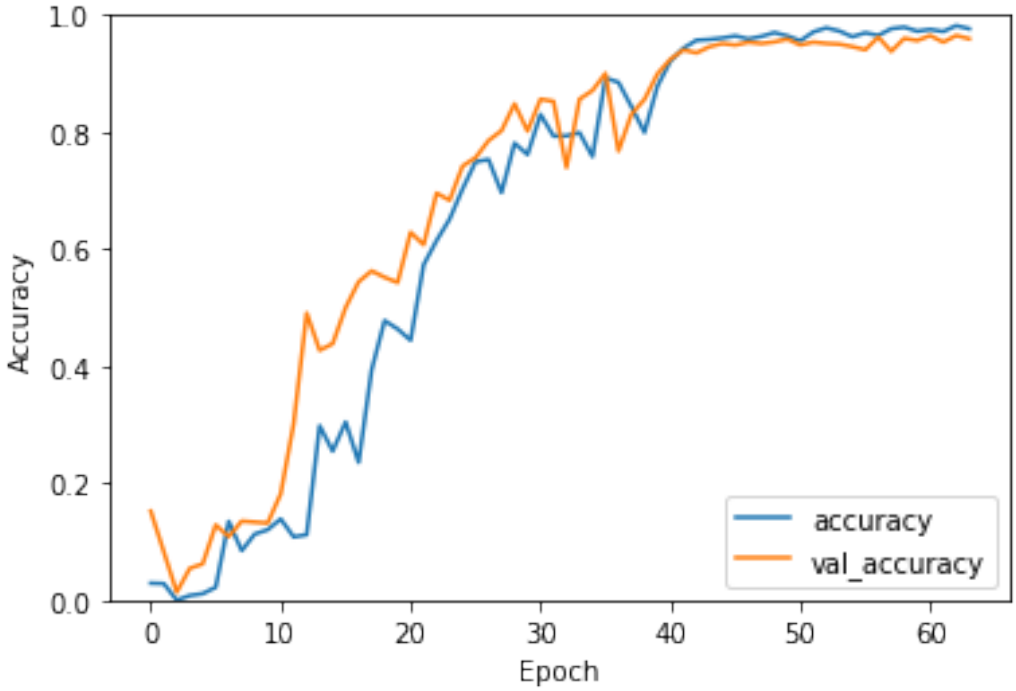

Figure 5.5: The training accuracy for the tri-headed model on mobiact

5.5 Additional Discussion

5.5.1 Improvement over Previous Models

We can see from the results that as the TE model architectures grow in complexity or design, the performance correspondingly increases. The base TE model is able to provide expectations on the kind of result we will observe on each type of data we feed in. From there, a direct improvement is made to the model and the first problem is identified: the cost of processing sequences. To reiterate our conclusion from the Improved TE section, subsequent tests with various processed MobiAct datasets aside from the Full Sequence data end up consuming too many resource and is therefore deemed ineffective. However, we do observe an increase in performance for the Full Sequence MobiAct, which continues to motivate improvements on the TE design. Incorporation of an additional temporal dimension from Keras/TF's TimeDistributed layer achieves two milestones: 1) further improvement in classification accuracy and 2) decrease in processing time per sequence. From here, we are able to utilize a plethora of datasets to test the limits of the TE model. Additionally, clustering using the extracted temporal features begin with this iteration. The last evolution we explore is multi-dimensional feature analysis using multiple CNN heads. While the classification performance saw minimal improvements, the clustering metrics return far greater scores than previously seen. The question now is whether the increased resource consumption of the Tri-Head model is worth the additional boost in clustering performance.

5.5.2 Cost vs Performance

Since our motivation is to deploy the pipeline for real-time clustering, leveraging model cost versus performance is an important topic to address. Relating to this, a thought that is raised from our experiments directly link to the near perfect scores achieved using the Full Sequence MobiAct data. From the TimeDistributed models onwards, we consistently observe the highest accuracy and clustering scores on that dataset. We explained our thoughts briefly throughout the experimental sections but to re-iterate, we believe that giving the LSTM models greater access to further periods of time allows the model to learn the specific intricacies for each activity. What we are essentially doing when we create an observation with a single full sequence of one action is forcing the model to learn time dependencies across all the timesteps. We believe the increased performance is also due to the replacement of LSTM layers with BiLSTMs, meaning the model has access to both past and future timesteps. The main problem with this is that the Full Sequence MobiAct dataset is not entirely realistic. Let us say we deployed the model trained on this dataset onto some live-stream processing system. If the system requires a prediction every second, there is no guarantee our train model can return the correct answer since the sequences it was trained on consists of data recorded for up to 5 minutes long. The ideal scenario would be to emulate the performance of models trained on the Full Sequence data but using more realistically segmented data sequences.

The immediate and obvious idea is to increase the window size. In the Full Sequence data, the window size could technically be considered the entire activity file, since the window size determines how many timesteps make up a sequence. If our default 200 (1 second) is seemingly not as effective and the entire activity file is

too effective but unrealistic, could we not choose a larger window size and increase performance? The thought is, the more timesteps the model can create temporal dependencies for, the better the resulting features should be. Our subsequent test with a window size of 512 timesteps (2.56 seconds) returns around the same classification accuracy using the TimeDistributed architecture (93.8%). However, notable improvement is seen during clustering, with K-means giving NMI and ARI scores of 0.82 and 0.579, respectively, and 0.681 NMI and 0.540 ARI when using FISHDBC. Seemingly, the larger the window size, the better the clustering performance but a later experiment with a window size of 1024 shows this is not exactly true. Referencing Table 5.10, we can see that the clustering score for 1024 suffers in quality compared to previous iterations. Except, the next experiment with using a window of 2048 sees a large leap in classification accuracy and clustering performance. On the surface, this result proves the longer the window the better the performance, but breaking down the actual classes returned with a window size of 2048 shows only 5 activities - the largest five. Diving into the data sequences provides an explanation as to why the classes reduced from 16 to a mere 5 (Table 5.12). Some activity files do not even have 2048 data points, so those were removed from the training data.

In addition, since we take the maximum frequency activity as the the determining action for that window sequence, the naturally high sequence actions (such as WAL or STD) overwhelms the short and sporadic activities such as the falling ones. As for why the window of 1024 returns poorer performance, one explanation could be using a window size of 1024 just happens to create sequences that cuts off activities. For example, if the full sequence consists of 800 walking timesteps, 300 falling timesteps, and the 400 lying down timesteps, a window of this size will cut off the sequence in the

middle of the falling points. The model therefore learns that this strange mixture of points corresponds to a walking sequence. The next sequence begins halfway through falling and ends with lying down and whatever action follows. The model will learn to classify this mixture based on the greatest frequency of actions. An additional useful trend we noticed is that the time taken to process a 128 batch of activities decreases as window size increases Table 5.11.

Table 5.10: Results from varying window size.

Window Size	**Classification**	**K-means**	**FISHDBC**
200 (default)	93.8%	0.664 NMI 0.628 ARI	0.564 NMI 0.372 ARI
512	93.8%	0.821 NMI 0.579 ARI	0.681 NMI 0.540 ARI
1024	93.2%	0.799 NMI 0.559 ARI	0.558 NMI 0.330 ARI
2048	**98.1%**	0.923 NMI 0.897 ARI	0.714 NMI 0.679 ARI
Full Seq.	**98.2%**	0.99 NMI 0.99 ARI	0.98 NMI 0.98 ARI

Table 5.11: Time taken for each 128 batch.

Window Size	**Time (seconds)**
200	21
512	17
1024	15
2048	11
Full Seq	3-5

While experimental results does seem to show that the longest window size improves performance, the solution is not to simply make the window size as large as possible since situations such as with windows 1024 and 2048 could occur. If the

Table 5.12: The number of activities returned for each window size.

Window Size	Activities Returned
200	16
512	16
1024	15
2048	5
Full Seq	16

window length is dynamically set to the length of the activity file, then the previous issue of impracticality remains relevant. On the other hand, if we set the size to a large number (such as 1024 or 2048), there is the chance that the window will segments sequences unnaturally or be overwhelmed by the highest frequency actions and prevent shorter and instantaneous movements from being registered. In the situation of the window size being longer than the entire sequence, that sequence would be disregarded and we lose information to learn. Therefore, a balance between the full sequence and too small of a window must be made through expert knowledge of the dataset to effectively learn HAR features and deploy onto real-life data streams.

Tables 5.13 and 5.14 present a synthesis of the core experimental results.

5.6 Summary

In this chapter, we applied the concept of Temporal Extraction in hopes of improving the quality of features extracted from HAR data for use towards clustering. Initial experiments on the various TE architectures show continual improvement in both classification and clustering performance as the models become increasingly complex. However, the returned clustering results were not ideal despite the high classification accuracy - until a particular method of data processing used with the MobiAct data

achieved near perfect scores for both classification and clustering. Subsequently, further experimentation into the reason for the marked increase led us to believe that the window length designated for each sequence plays a vital role in how much temporal dependencies the TE models are create between observations. While a full sequence window size shows the best performance, the relationship between window size and performance is not a linear one, as experiments with certain window sizes have shown to cause additional problems such as dropped classes or lowered accuracies as a result of unintentional bisection in a sequence of actions.

Table 5.13: Summary of experimental results

Algorithm	Dataset	Classification	Clustering	
		ACC	ARI	NMI
Stand-alone Clustering Algorithms				
FISHDBC	MobiAct SLH	-	0.348	0.535
WCDS	MobiAct SLH	-	0.497	0.568
DEC	MobiAct SLH	-	0.406	0.498
Autoencoders				
DEC	MobiAct	29%	0.24	0.32
	UCI	49%	0.49	0.54
	MNIST	91%	0.91	0.91
2D CNN	MobiAct	93-99%	0.441	0.594
2D CNN + AE			0.340	0.555
LSTM AE	MobiAct	89%	0.344	0.598
	UCI	95%	0.461	0.613
Hybrid AE	MobiAct	94%	K-means	
			0.719	0.856
			FISHDBC	
			0.678-0.832	0.743-0.808
	UCI	95%	K-means	
			0.99	0.99
			FISHDBC	
			0.731-0.796	0.870-0.912
	12 Class MobiAct	95%	K-means	
			0.831	0.897
			FISHDBC	
			0.86	0.88

Table 5.14: Continuation of Table 5.13

Algorithm	Dataset	Classification	Clustering	
		ACC	ARI	NMI
Temporal Feature Extraction				
Base TE	Many-to-One MobiAct	92.6%	-	-
	Seq2Seq MobiAct	92.7%	-	-
	UCI	92.4%	-	-
Improved TE	Full MobiAct	95%	-	-
TimeDistr. TE	Raw Reduced MobiAct	93.1%	K-means	
			0.628	0.664
			FISHDBC	
			0.372	0.564
	Engineered Reduced MobiAct	93.6%	K-means	
			0.431	0.649
			FISHDBC	
			0.348	0.543
	Full MobiAct	98%	K-means	
			0.99	0.99
			FISHDBC	
			0.98	0.98
	UCI	92.2%	K-means	
			0.927	0.929
			FISHDBC	
			0.875	0.860
Tri-Head TE	Raw Reduced MobiAct	93.8%	K-means	
			0.522	0.749
			FISHDBC	
			0.648	0.634
	Engineered Reduced MobiAct	93.8%	K-means	
			0.548	0.784
			FISHDBC	
			0.673	0.684
	Full MobiAct	98%	K-means	
			0.99	0.99
			FISHDBC	
			0.98	0.98
	UCI	93.5%	K-means	
			0.99	0.99
			FISHDBC	
			0.93	0.94
	512 window MobiAct	93.8%	K-means	
			0.579	0.821
			FISHDBC	
			0.540	0.681

Chapter 6

Visualization of High-Dimensional Clusters

The ubiquity of high-dimensional data at nearly every sector of research presents a global problem in comprehension of the information within the vast array of numbers. The large collection of features reported by these datasets have become conceptually unimaginable as the dimensions exceed two or three [80]. Visualization without prior processing creates graphs of indecipherable nature - if the plot is even successfully generated. Fundamentally, the problem is humans can not view beyond three dimensions, so visual analysis of higher dimensional datasets must be reduced to a level within our perception, and even then intuitive knowledge must be derivable from the representations. Dimensionality reduction is the technique used to do exactly that - it transforms data from high dimensions to low dimensions while still retaining particular properties of the full data. Visualization methods can be applied on to the reduced representations to provide a medium where humans can procure knowledge about the dataset.

Human Activity Recognition (HAR) is the field of study aiming to identify the actions or movements of an agent using observations of the subject or its environment. Since research in this area involves uncovering the nature of human movements, the

datasets generated for this field are complex and prone to variances - values that are both reflected in their subject. The volatility of HAR data implies a level of uncertainty that naturally exists when experimenting. Being able to understand at a deeper level the patterns and information contained within the dataset would provide an invaluable amount of knowledge to invest towards model design and interpretation of the test results. However, the benefits gained from visualizing a dataset does not only occur prior to processing. When it comes to analytical methods such as clustering, investigating a visual representation of the clusters returned by the algorithm can grant precious insights into the distribution of classes or the quality of the clusters. The two metrics globally used for clustering, NMI and ARI, provides a simple evaluation on the performance of the clustering model, but it does not provide any deeper knowledge aside from this. Manually validating the aftermath of clustering can yield valuable intuition into the dataset.

Over the years, various works have provided surveys on the high-dimensional data visualization domain. Bertini et al [15] presented a systematization of visualization metrics for quality validation of low-dimensional projections. The paper of Heinrich et al [52] contributed a collection of algorithms centered around parallel coordinates while the literature published by Ellis et al [36] focused on clutter reduction in the resulting low-dimensional representations. A survey on multifaceted high-dimensional data was prepared by Kehrer et al [62], covering multiple visual analysis methods. Finally, a thorough study of advances in visualization methods was arranged by Liu et al [80]. The paper provided a comprehensive analysis of algorithms in the past decade and expanded upon all manner of sub-domains related to high-dimensional visualization.

This chapter will provide an analysis of older, dependable dimensionality reuction algorithms in comparison with modern state-of-the-art reduction and visualization methods in reference to complex HAR data. The context for visualization is determining the separability of classes for future clustering, therefore distinct clusters are the ideal results, with interpretability of class interactions also valued substantially. We will display plots of the reduced representations and visually validate the graphs using expert knowledge of the dataset to show that modern dimensionality reduction and visualization methods have improved on the older generational algorithms and are capable of providing higher levels of interpretability and intuitive understanding of the dataset under research.

The rest of the chapter is organized as follows: Section 6.1 will be a literature review covering notable visualization algorithms. The dataset we used for this chapter is covered in Section 6.2. Section 6.3 explains the groundwork reduction algorithms and applies them onto a subset of the MobiAct data for visual example. Modern methods are presented in Section 6.4, where both the subsets of the MobiAct data and the clustering results from Chapter 5 are visualized. Finally, Section 6.5 summarizes this chapter.

6.1 Literature Review

The literature for this domain is vast and encompassing, with published papers covering topics convoluted enough to be their own separate discipline [80]. Our review will primarily focus on select pieces of dimensional reduction algorithms or frameworks for purposes of visualization. Most of these works will include their own visual methods. Later in the chapter, we will present background on foundational algorithms, and any

direct literature improvements on those algorithms will be listed in their respective section.

Moon et al [92] presented an algorithm to address several problems they identified in the domain: performance trade-offs between local and global structures presentations, noisy visualization, and lack of scalability. Potential of heat diffusion for affinity-based transition embedding (PHATE) utilizes diffusion geometry [92] to consider both local and global densities when creating the embeddings. The representation is passed through a filter using the diffusion technique and the result is low-dimensional projection optimized for visualization. Additionally, the authors formulated a new metric that evaluates based on preservation of relationships and invariability to noise.

The LargeVis method was proposed by Tang et al [114] as a successor to t-SNE. The algorithm approximates and improves on the construction of the k-nearest neighbour graph by implementing projection trees [31]. The approach builds a tree by randomly sampling two points until a hyperplane of equal distance between the points is found. This hyperplane is split and the corresponding subspaces becomes that node's children. However, this method requires multiple runs to converge onto an optimal solution. The authors present a solution by constructing a low-accuracy approximation and deploying neighbour exploring techniques. The ideology is that the neighbouring node of a neighbour has a high chance of also being a neighbour to the current node [35]. These methods allowed LargeVis to either greatly outperform or completely match t-SNE.

A 2018 paper takes a different approach by using neural networks for dimensionality reduction. Ding et al [34] were motivated by relative ineffectiveness of t-SNE

when applied towards high-dimensional single-cell RNA data and created scvis to overcome the identified problems. Scvis is a feed forward neural network with the ability to learn a parametric mapping function from the high-dimensional data to a low-dimensional projection. The model is also probabilistic, therefore is capable of estimating the uncertainty of incoming new data when applied towards the learnt mapping function.

6.2 Dataset

The datasets we use for our experiments are the 2017 MobiAct v2.0 HAR collection [25] and the University of California Irvine HAR dataset [7] from their machine learning repository. The exact details of the datasets are described in Chapter 3, so we will refrain from reiterating that information here. For Sections 6.3 and 6.4, we use a subset of the Scenario of Daily Living data namely the 'Scenario Leaving Home (SLH)' to provide visual examples of the algorithms. This is the same scenario we used in Chapter 3 to evaluate the FISHDBC, WCDS, and DEC algorithms. We also visualize the clusters created from the full MobiAct dataset and the UCI collection, but will limit their use with the state-of-the-art visualization algorithms.

6.3 Dimensionality Reductions

PCA (Principal Component Analysis) [102] and t-SNE (t-distributed Stochastic Neighbour Embedding) [120] are two foundational methods for dimensional reduction and are considered fundamental algorithms for all areas of research. Both algorithms map data from high-dimensions to lower-dimensions, but achieve it using different techniques. The details will be expanded in their respective sections.

In order to compare the performances between the old and the modern techniques, we will apply them towards the MobiAct SLH data and establish a baseline using the recognized algorithms. 2-Dimensional and 3-Dimensional plots of the reduced data generated by the benchmark PCA and t-SNE methods will provide a reference for the modern dimensionality reduction iterations.

PCA

An intuitive understanding of PCA can be imagined by trying to summarize an extensive list of wines. Each bottle has multiple properties but not all are relevant, so PCA conjures up a new set of characteristics using a combination of the old ones. The generated features contains enough information (variance, in this case) to ideally reconstruct the original characteristics of the wines. Fundamentally, that is all the algorithm is doing - reducing complex features into a representational set of lower dimensional data. For a more complex explanation, PCA is a orthogonal linear transformation method that maps data to a new coordinate system. The algorithm works such that the first coordinate (or component) contains the largest variance from the projected data, and each subsequent coordinate contains data with the next largest variance. PCA works to find the perfect linear combination such that the maximum variance and minimum reconstruction error is reached for those projected features.

```
Explained variation per principapal component:
[0.83940311 0.15475138 0.0027412 ]
```

Figure 6.1: The variance explained by each component after performing PCA reduction on the MobiAct scenario sub-dataset.

Fig 6.1 shows the variance of the 3 components after applying PCA onto the

MobiAct scenario sub-dataset. As seen, components 1 and 2 account for the majority of the variance, meaning the two features best explain the distribution of the projected data. This also implies that the data can be successfully modeled using only two dimensions without losing substantial information.

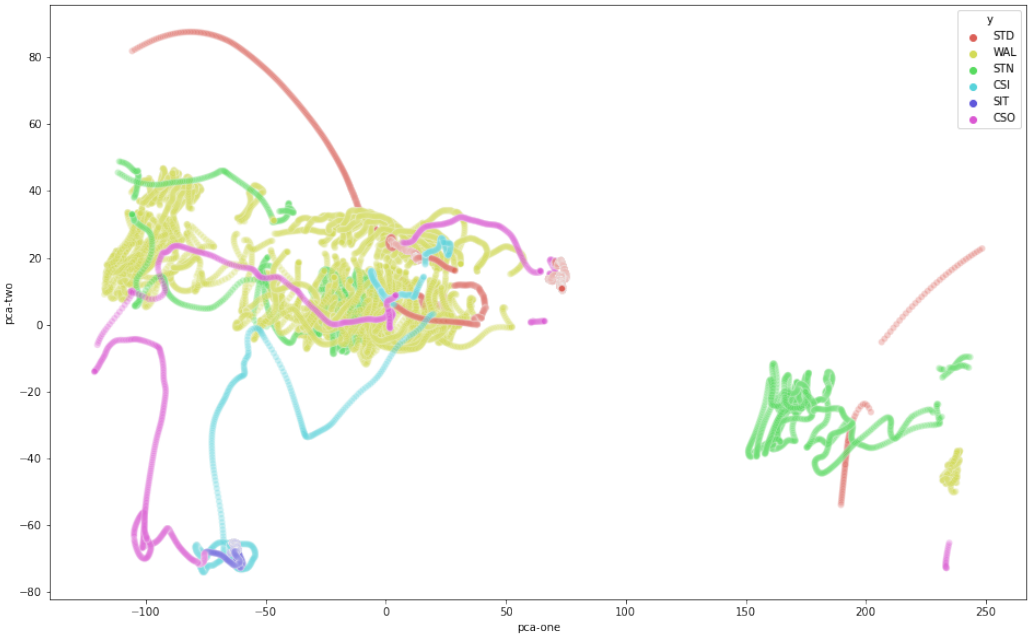

Figure 6.2: Plot of the two highest variance components after PCA reduction. In this case, pca-1 and pca-2 were used as the axis.

Figure 6.2 is the 2D plot of the subset after applying the PCA algorithm. One obvious distinction we can see is the 'path' of activities and how they flow into each other. Most notably the segment on the lower left visualizes the transition from the CSI (Car-Sit-In) activity into SIT, and subsequently into a CSO (Car-Sit-On) sequence. This example highlights the ambiguous boundary between activities and provides some perspective into difficulties some models might encounter analysing complex data like this. Another notable observation is the separate cluster on the right belonging to STN (Stairs Down). Presumably, the walking motion in combination

with the downwards momentum from descending the stairs is highly distinguishable from the other relatively flat motions of the other activities. The only other actions that display this kind variance in verticality are the CSO and CSI activities, which can be observed as relatively distinct. A frequent pattern we will detect throughout the visualization graphs is the encompassing cloud of data belonging to the WAL (Walking) activity. The density of the activity in the dataset accounts for the mass of points, but the interesting observation is the intertwining nature of adjoining activities and the variability of the walking sensor values. The figure presents an intuitive explanation as to why the stand-alone clustering algorithms had difficulty processing the 6-class SLH sub-dataset. The central mass of pink (WAL) points are mixed in with a number of other activities, making it difficult to differentiate between the data points. As far as the clustering models are concerned, the entire clump of data counts as a single WAL cluster. For any analytical model to be successful in clustering this data, a deeper representation of the data distribution must be learned to 'untangle' the activities from each other. The intertwining phenomenon is most likely representative of the transitional nature of human activities - one action does not just end, it leads into further actions.

Despite PCA reducing the data perfectly into two major components, we visualized the data in 3-dimensions to gain a difference perspective. An additional PCA component was created and used as the third axis. Figure 6.3 is the resulting graph. We observe the general shapes and clusters seen in Fig. 6.2 but with additional directionality. No deeper intuition can be extracted as the third PCA component contains near negligible variance (and therefore, information).

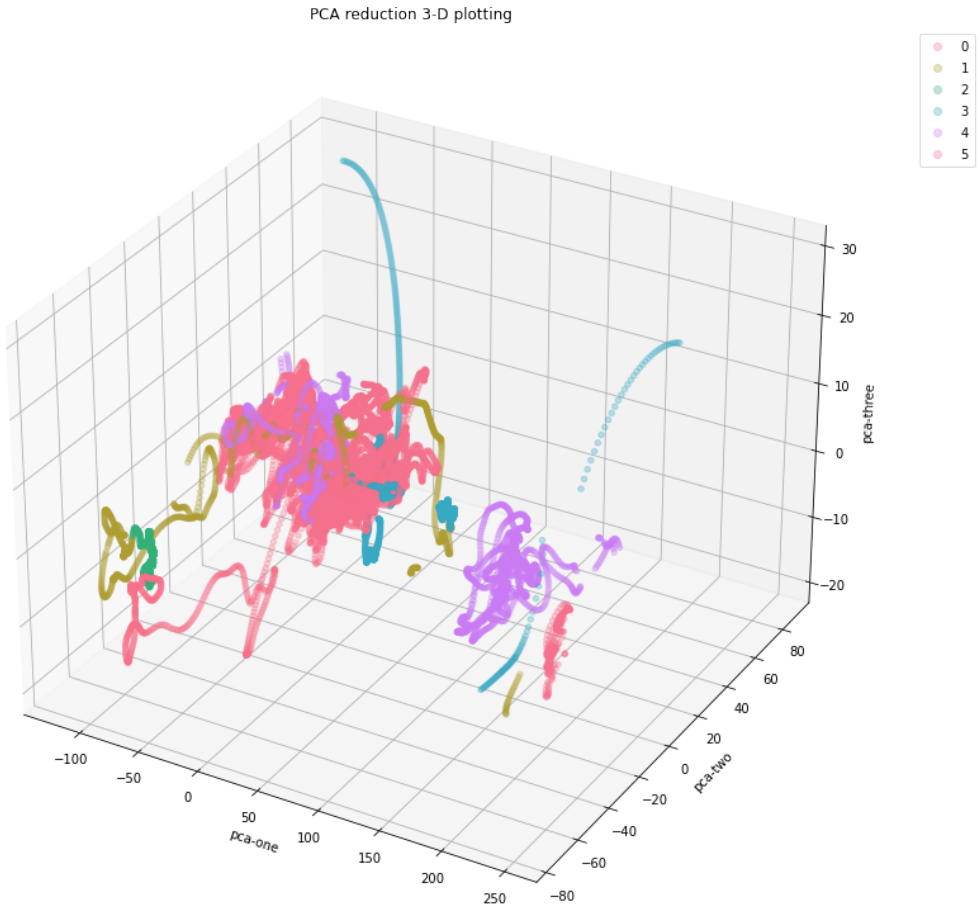

Figure 6.3: 3-D plot of the PCA reduction on the MobiAct SLH scenario

t-SNE

T-SNE [120] is another foundational algorithm that is considered the standard for dimensional reduction and subsequent visualization. It is a statistical nonlinear technique that works by comparing probability distributions mapped over the high dimensional data and low dimensional embeddings in order to minimize the Kullback-Leibler (KL) divergence (Eq. 6.1). The KL Divergence [71] loss indicates the difference between the distributions, P and Q. If the probability of an event in P is large but small

in Q, the function returns a large score. Conversely, a smaller score is reported if the probability of the event in P is small but large in Q. The probabilities used by the function are assigned by t-SNE - similar objects are granted a higher probability while dissimilar ones receive a lower probability.

$$KL(P||Q) = \sum_{x \in X} P(x) * \log(\frac{P(X)}{Q(X)}) \tag{6.1}$$

T-SNE differs to PCA in their priority of data feature preservation. PCA tries to maintain large pairwise distance while maximizing variance - in this sense, dissimilar objects will be further apart. Conversely, t-SNE preserves small pairwise distances or local similarities. As such, t-SNE does not preserve attributes like density, making the reduced representation unsuitable for clustering and only functionally useful for dataset exploration. An additional difference is that the cost function of t-SNE is non-convex, meaning multiple instantiations can produce nonidentical results.

Figures 6.4 and 6.5 are the 2-dimensional plots produced using reduced data fed through the t-SNE algorithm. Perplexity is a measure of information defined as 2 to the power of the Shannon entropy [120], and can be intuitively thought of as the parameter controlling effective nearest neighbours. As seen in the images, the perplexity can greatly affect algorithm performance. A perplexity of 100 returns a sparse representation that conveys a summarization of the dataset, while the perplexity of 300 creates dense clusters expressing the separability between certain activities.

The same features observed in the PCA graphs can also be seen in the t-SNE plots, with the most obvious being the overwhelming mixture of various activities into the WAL cloud clusters. The intertwining phenomenon we observe in both the

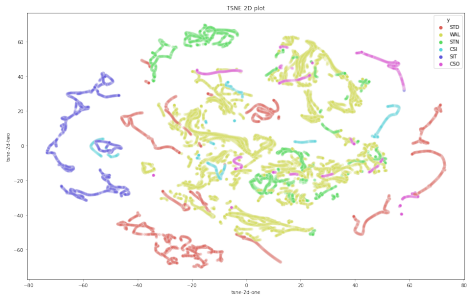

Figure 6.4: t-SNE with perplexity of 100

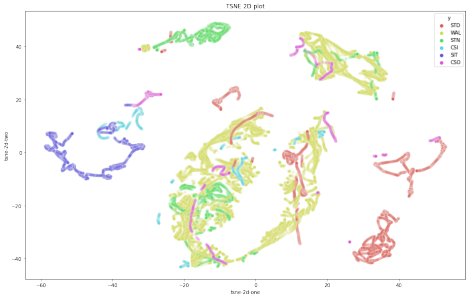

Figure 6.5: t-SNE with perplexity of 300

PCA and t-STN graphs further accentuates the difficulty algorithms have in differentiating activities belonging within these clusters. We can also notice the previously identified CSI-SIT-CSO combination re-emerging in the left side of Figure 6.5. Notably, certain activities are separated into multiple clusters. This can be explained by the conservation decisions of t-SNE. Since t-SNE preserves local similarities, the larger flow of movements were not captured as it was using PCA. Rather, the smaller variances were prioritized. The information extracted at the beginning of a movement were mostly likely too dissimilar to those recorded at the end, leading to the disjointed groups seen in the plots.

To gain a different perspective, the t-SNE was applied again but using 3 components to enable 3D plotting. Figure 6.6 shows the subsequent visualization of the data. An immediate distinction seen differing from the other graphs is the flow of movements captured by the plot. While the other graphs show separability, the 3-dimensional t-SNE graph displays the variability of the agent as it traverses through the sequence of actions. A clear path can be followed as the subject moves down the stairs, into their vehicle, and finally ending with the STD (standing) cluster at the bottom left. Figure 6.7 is the same plot but with individual activities separated

for clarity. Using the three graphs, we can still see the dense, highly variant WAL data cloud that interweaves with every other action. We can make out the closely related STN points located right in the middle of the WAL cluster. The reason why PCA was able to separate this from the WAL points entirely is probably because the overall distances calculated for the STN activity showed large variances; enough to distinguish from the WAL activity. t-SNE does not preserve the overall distances, so in terms of specific intervals, the STN activity looks very similar to the WAL activity.

Other interesting features are the SIT and STD clusters of Fig. 6.7. Despite the intuitive understanding that sitting and standing would produce the least variance in sensor data, the subsequent t-SNE reduction shows an amount of variability. The STD variance can be explained by the difference in elevation at the beginning of the scenario compared to the end, in addition to the natural body instabilities captured by the sensors. The SIT variances might be due to imprecise labelling by the authors of the dataset, thus allowing for excess variability. Another observation is the slight arch present in several of the activities. T-SNE managed to capture the difference in verticality as the subject moves through the scenario. For example, the WAL and STN plots in Fig 6.7 shows a slight curve representing the subject as they move down the stairs and continue walking. The STD plot shows two distinct groups of points - one group represents the subject standing on top of the stairs, and the second group represents the subject in front of their car.

T-SNE is, however, not without problems. Originally presented as a proof-of-concept algorithm [120], t-SNE has difficulty scaling to very large datasets. This can prove troublesome as a typical use case for visualization algorithms is towards high-dimensional biochemical data - datasets that can contain information on upwards

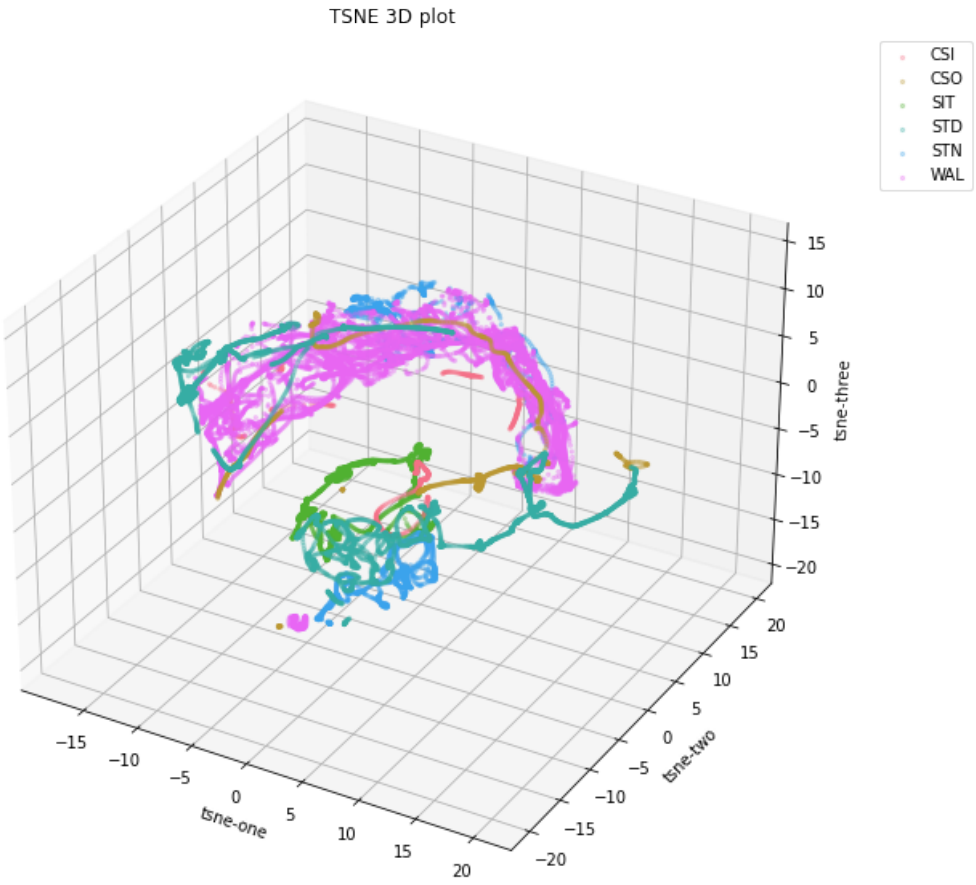

Figure 6.6: t-SNE 3D plot of all activities.

of 1.2 million cells [112]. Several works aimed to overcome this problem. In 2014, the original author published an implementation of t-SNE that utilizes a tree-based algorithm [119]. Named the Barnes-Hut variant, this version was able to scale the original algorithm to handle datasets with millions of objects. Inspired by this work, a GPU assisted adaptation was proposed in 2018 [24]. The model, named T-SNE-CUDA, leveraged the power of a GPU to achieve 50-700 times the speed of the best implementations at that time. Finally, a 2019 algorithm utilized fast Fournier transformations to greatly improve on the 2014 Barnes-Hut t-SNE variant [76].

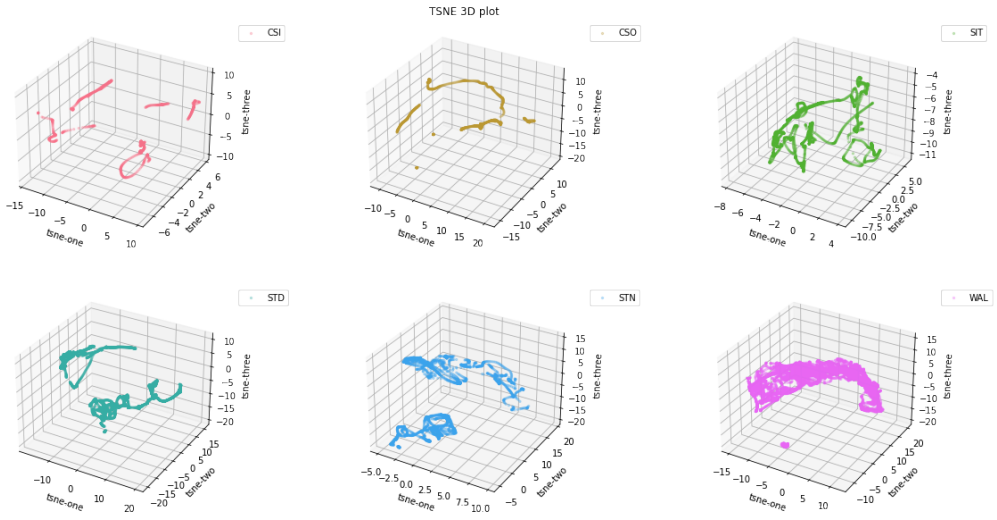

Figure 6.7: t-SNE 3D plots of the individual activities.

6.4 State-of-the-art Visualization Algorithms

The algorithms listed in this sections are all modern cutting-edge frameworks that improve on the PCA and t-SNE methods for feature reduction in some capacity. All these methods also visualize the projected data using their own methods. Additionally, we will apply the temporal-extracted clusters onto the SOTA visualization algorithms to provide intuition and manual analysis on the quality of feature extraction.

UMAP

UMAP (Uniform Manifold Approximation and Projection for Dimension Reduction) [90] is a 2018 algorithm created with the purpose of succeeding the state-of-the-art benchmark visualization method of the time: the t-SNE algorithm. The algorithm essentially constructs and performs operations on weighted graphs. In a practical sense, the algorithm works similarly to k-nearest neighbour based graph algorithms. UMAP

first constructs a weighted k-neighbour graph, then a low dimensional representation of the graph is created. From a higher theoretical perspective, UMAP approximates a low dimensional manifold (a topological space) of the high level data to create a topological representation. Another topological representation is manufactured using a low dimensional representation of the data. The algorithm then tries to minimize the cross-entropy of these two topological representations, similar to how t-SNE tries to minimize KL divergence.

Figure 6.8 and Figure 6.9 shows the returned representation of the MobiAct SLH sub-dataset using different parameters (n_neighbours 300 and 100, respectively). These plots exhibit patterns observed in the other graphs: the CSI-SIT-CSO sequence clearly flows into each other; the WAL cluster(s) encompasses a number of other activities; and clear distinctions can be made for certain actions such as STN and STD. The n_neighbour parameter affects algorithm performance and is reflected in cluster proximities shown in the plots. A n_neighbour value of 100 provides a general summary of the data, similar to t-SNE. The difference is that UMAP's representation is more accurate than t-SNE's projection in terms of overall structure. A parameter value of 300 distinguished identical activities into separate clusters - presumably because of variances in the raw values. The shape of the clusters in both graphs are similar, but the value of 100 returns the representation most accurate to the true sequence while the value of 300 creates more confident clusters. This can be seen in the 3 distinct clusters for WAL in Fig. 6.8 versus the single connected cluster in Fig. 6.9.

Applying UMAP to the full MobiAct and UCI datasets returns the graphs in 6.10 and 6.11. The immediate observation are the very distinct clusters for the large

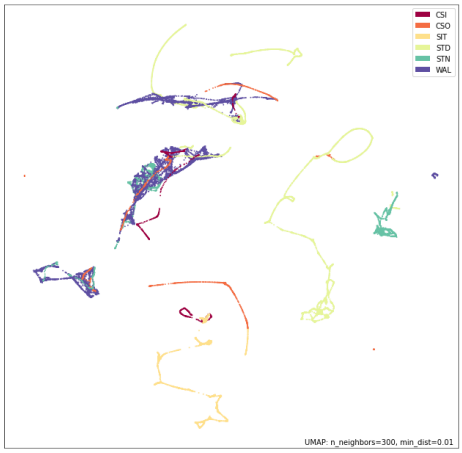

Figure 6.8: UMAP with n_neighbours of 300

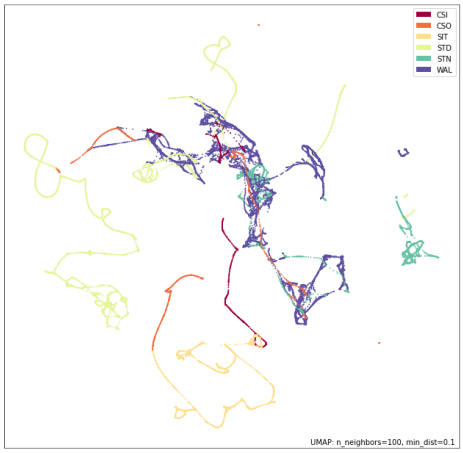

Figure 6.9: UMAP with n_neighbours of 100

activities (0: STD and 1: WAL) as well as the medium-sized mixed cluster of activities 7 (SIT), 8 (CSI) and 9 (CSO) on the right side of the MobiAct plot. The same pattern was observed in the t-SNE and PCA plots of the SLH sub-data, which contains the same combination of sequences. However, the clusters returned from the temporally-reduced data creates more defined boundaries and higher density shapes. The UCI plot is distinct and clear, reflecting the high clustering score achieved by the clustering algorithms.

DensMAP

Recently, an evolution to UMAP was proposed in a 2021 paper. Titled DensMAP [95], this algorithm takes everything that makes UMAP strong and improves on it by preserving local density information. This feature allows for a greater and more accurate understanding of the data distribution. DensMAP achieves this by using the same weighted nearest-neighbour graph and calculating the local radius around each point - this conveys information about the density of the neighbourhood around the

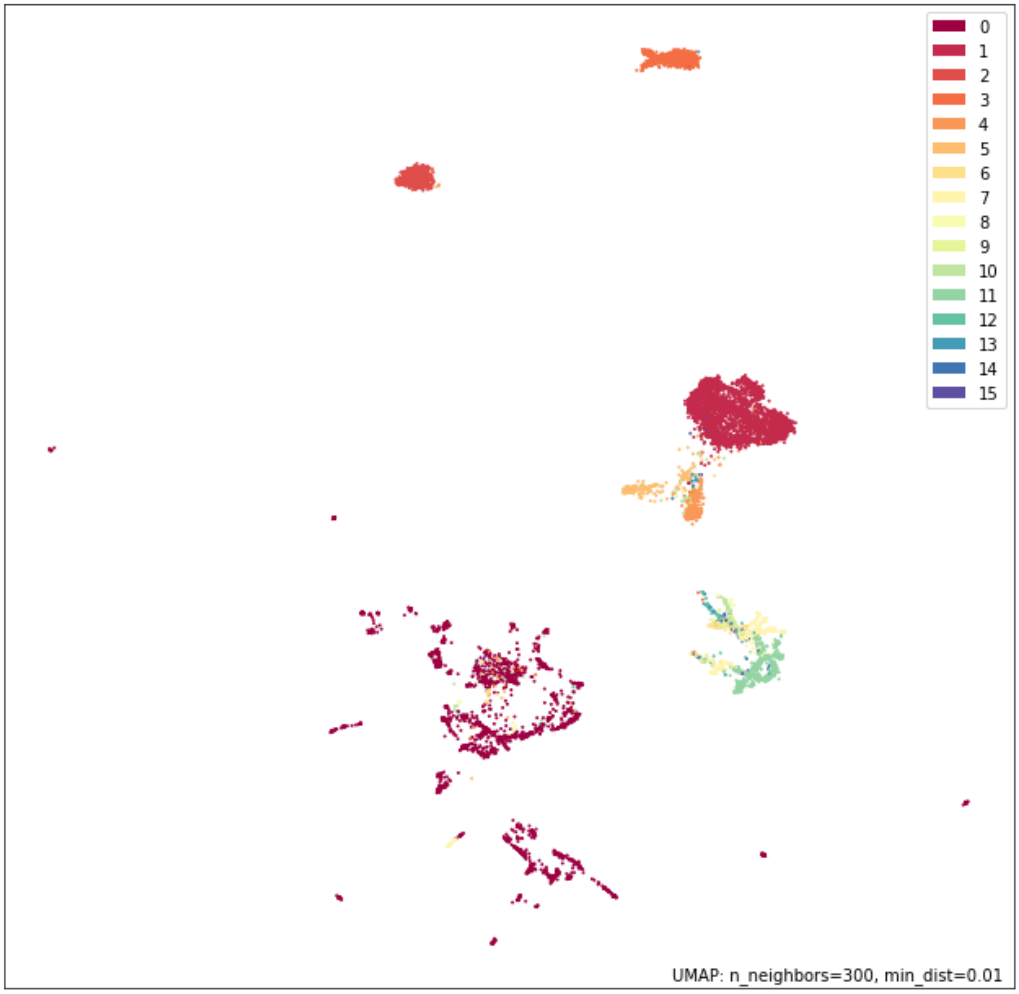

Figure 6.10: UMAP on the raw MobiAct dataset

point. Figure 6.12 is one of the representations returned by DensMAP when applied onto the scenario dataset. The improvements made towards local density preservation in DensMAP creates clusters reflecting more accurate pairwise distances compare to the previous DensMAP plots. This density-preserving feature is an improvement over t-SNE, and when combined with the efficient dimensional reduction methods of UMAP, implies that DensMAP can effectively replace t-SNE as a standard in the field. Maintaining better local density also means that we can deploy a density or

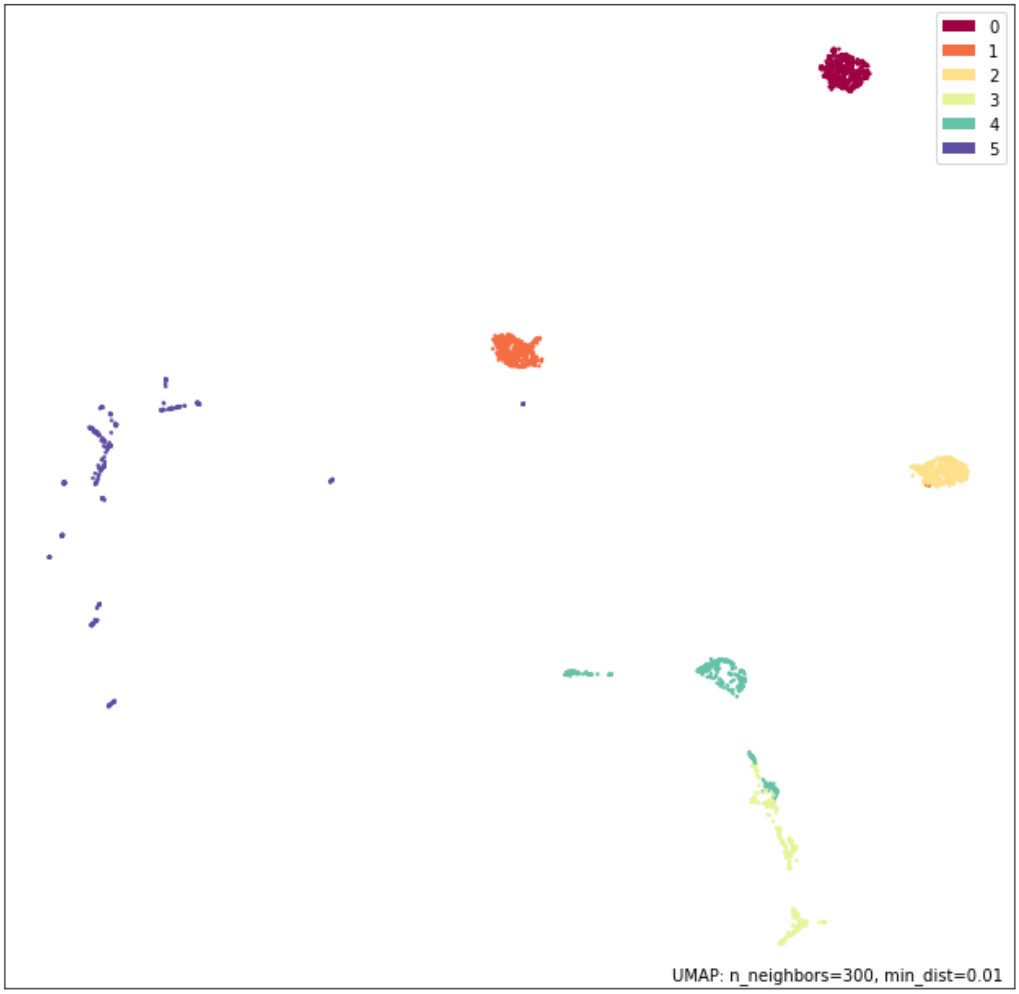

Figure 6.11: UMAP on the UCI dataset

distance-based clustering algorithm on the reduced dataset and still return clusters that are representative of the original data distribution. Fig. 6.13 and Fig. 6.14 provides a side-by-side view of the regular UMAP plot using the DensMAP.

The following graphs are the DensMap plots of the temporally-reduced MobiAct data (Fig. 6.15) and the UCI dataset (Fig. 6.16). As with the other DensMAP graphs, the same distribution used in UMAP is captured by the DensMAP but density is accounted for, therefore certain clusters will be more compact while others have

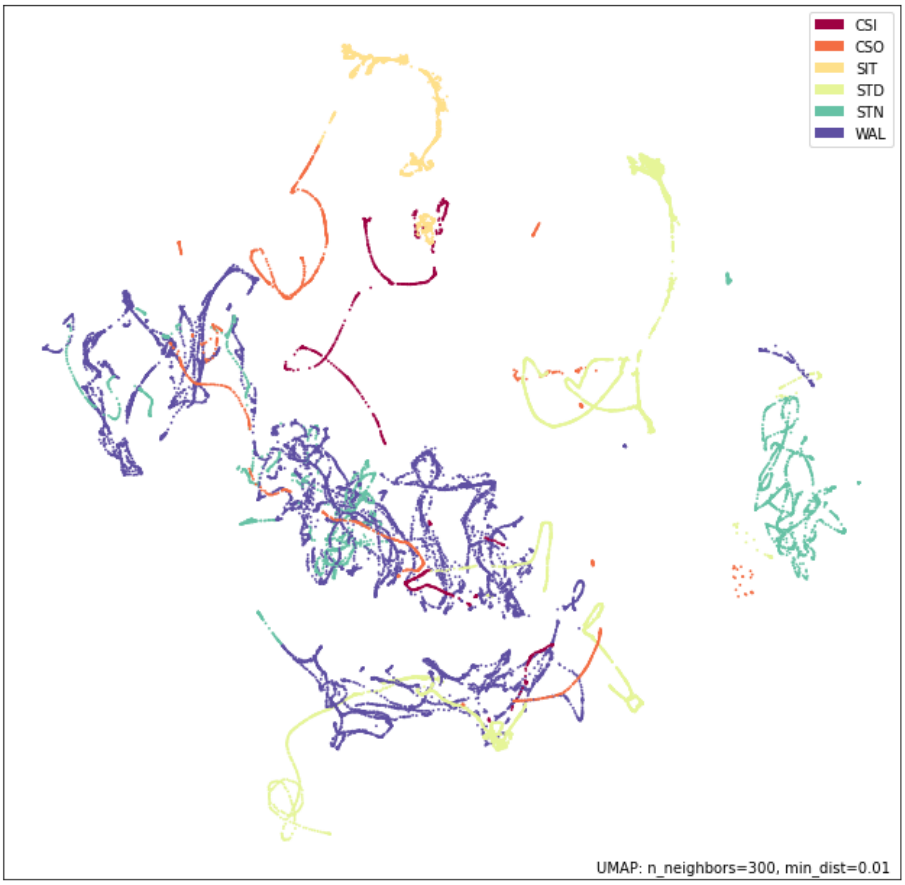

Figure 6.12: DensMAP with n_neighbours at 300

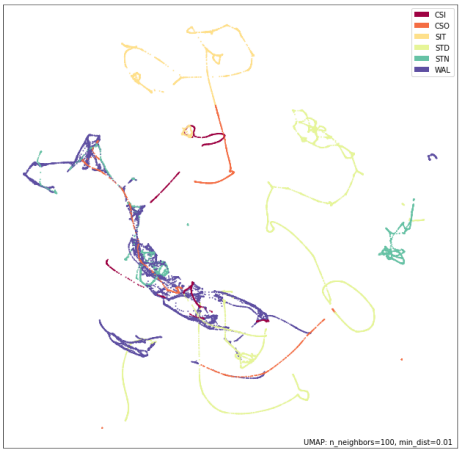

Figure 6.13: UMAP

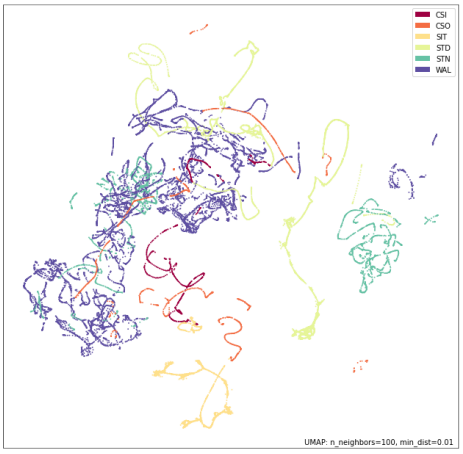

Figure 6.14: DensMAP

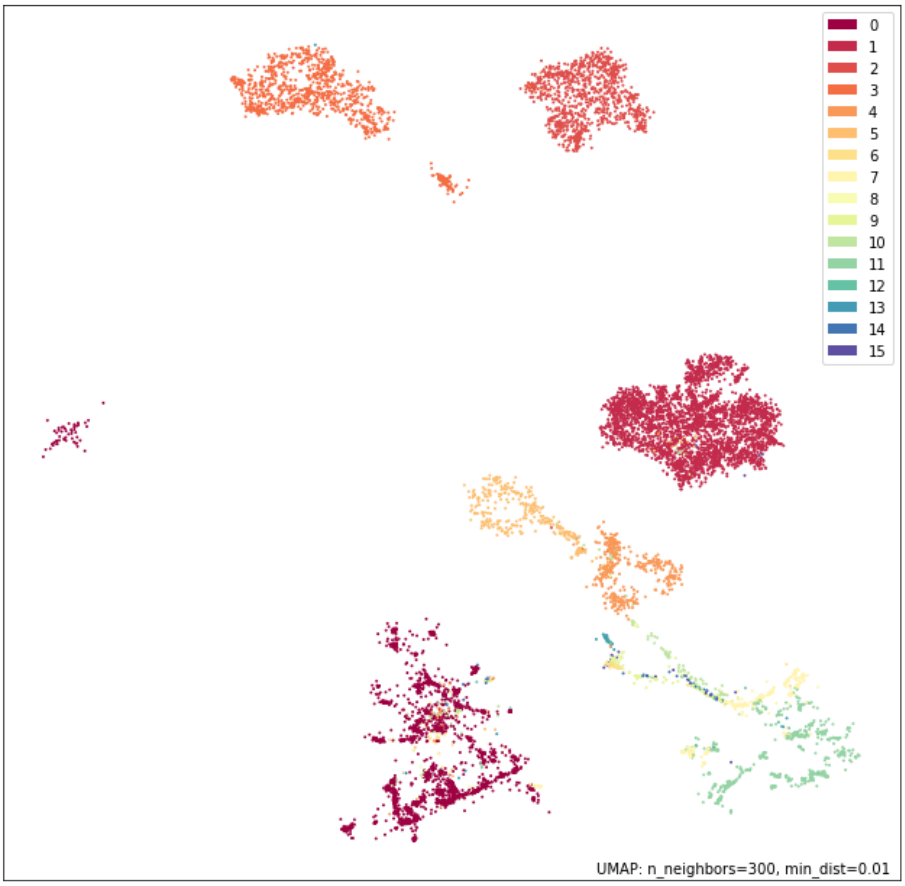

Figure 6.15: DensMAP plot on the raw MobiAct dataset

shifted in position in reference to other clusters. Ultimately, the same information is presented and the same intuitions can be derived from the DensMAP plots.

TMAP

TMAP (Tree MAP) [104] is a modern contribution in the high-dimensional reduction and data visualization domain. The authors proposed the algorithm in 2020 for the purpose of displaying extremely large dimensional data commonly found in molecular chemistry studies. They were motivated by a need to effectively and clearly display

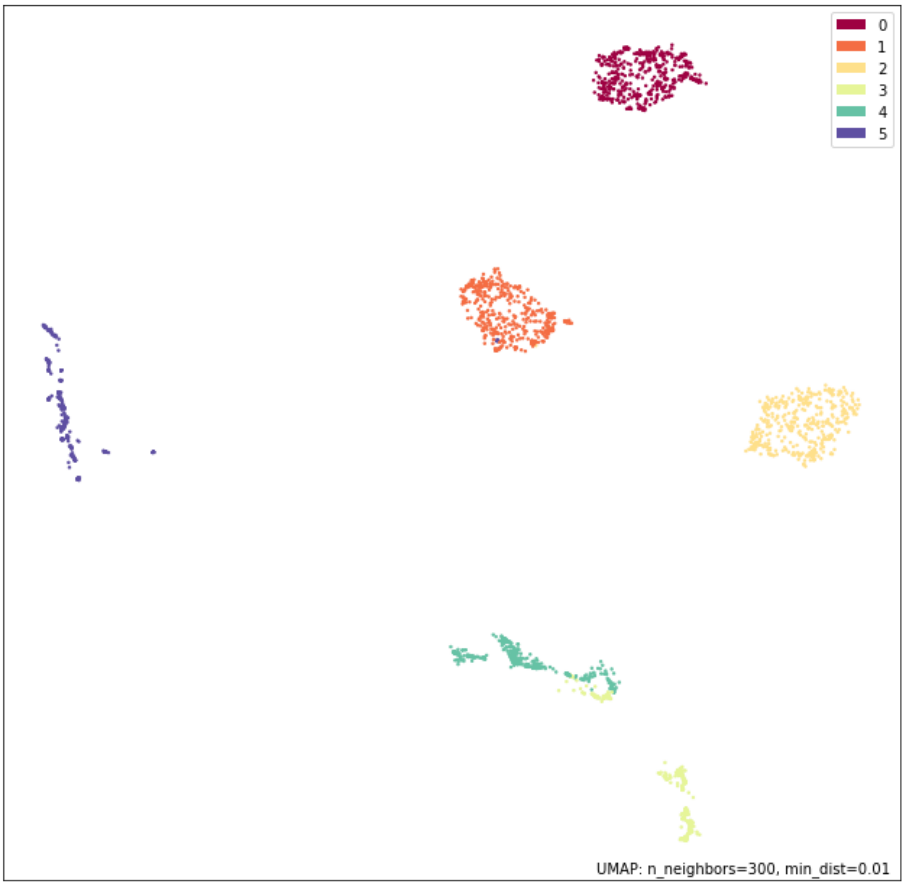

Figure 6.16: DensMAP plot on the UCI dataset

the relationships between complex features in a method that is still interpretable. As seen with the previous visualizations, some of the graphs returned by the algorithms can be difficult to interpret or derive meaning from. TMAP proposes a solution to this problem in the form of connected tree.

The TMAP algorithm, like the other methods listed here, also approximates a k-nearest neighbour graph. The difference is, this algorithm evolves the graph into a minimum spanning tree (MST). Any cycles in the graph are pruned, effectively lowering computational complexity for low-dimensional embeddings. To reduce effort

required to graph potentially millions of vertices in the tree, another approximation-based method is used to draw the MST. The result is an aesthetically pleasing graph of the low-dimensional representation of the dataset. Figure 6.17 shows an example of the MST created by TMAP and how the algorithm re-configures the information for display. Figure 6.18 is a plot of the MNIST handwritten dataset after applying TMAP's reduction and plotting onto the data. An interactive webpage is returned and allows for exploration of the data. The MNIST digits are clearly very separable, and TMAP creates an informative structure for intuitive analysis. We can easily discern the types of digits that blurs the algorithm's confidence by searching for outlying coloured points within each cluster.

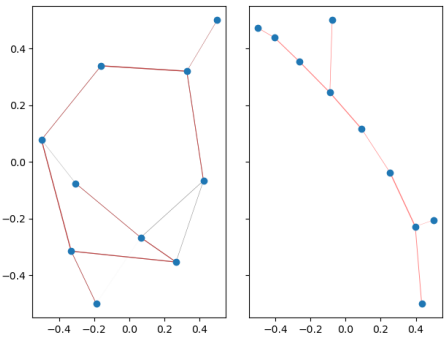

Figure 6.17: The tree structure created and idealized by TMAP [104]

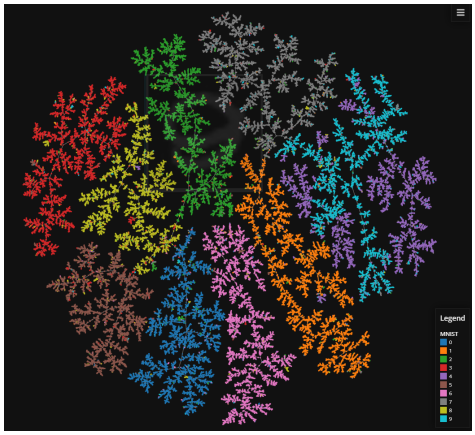

Figure 6.18: MNIST digits returned by TMAP [104]

Figure 6.19 shows the results of TMAP on the MobiAct SLH sub-dataset and Figure 6.20 is the TMAP result on the same dataset but after temporal processing using the Tri-Head model. An immediate observation in the full TMAP graph is the vast number of inter-twining orange WAL points with various other activities. This trend has been observed in every visual graph thus far, but the difference is that

TMAP provides the opportunity for a detailed exploration of the individual points. For example, we can trace the original values as 'walking' (orange) transitions into 'stair down' (green) and analyze the boundary points to identify which range of values determine the activity. The CSI-SIT-CSO sequence is still present in this representation, albeit more spread out. This view of the data also brings back a phenomenon we have mentioned before: the variability of the SIT and STD actions. The purple (SIT) and blue (STD) show tight clusters that split off into other similarly dense groups, something observed once before with t-SNE. Since t-SNE does not preserve much local density, we assume the sensor resolution is sensitive enough to capture minute movements of the subject and these are small enough to be considered highly similar, yet any change in agent positioning registers a transition large enough to be considered relatively different to the minute measurements. This can explain the disconnected phenomenon observed here. The chaos we observe in the full TMAP graph helps explain the poor performance by the standalone clustering models on the SLH sub-dataset. Human movements are complex and therefore, it is expected that the visualization will not be perfectly interpretable, however TMAP provides an opportunity to explore the specifics in interpretability.

The temporally-processed TMAP plot represents very clear clusters and separations between the activities. Intuition can be derived from the image easily because of the simplistic representation. For example, the light pink line of SIT points are very clearly connected and associated with the brown (CSI) and light brown (CSO) points. This is as expected just from intuition. An interesting relationship are the STD (blue) points and the light green STN points - they are shown to be related but intuitively they are quite different since the STD action is practically motionless

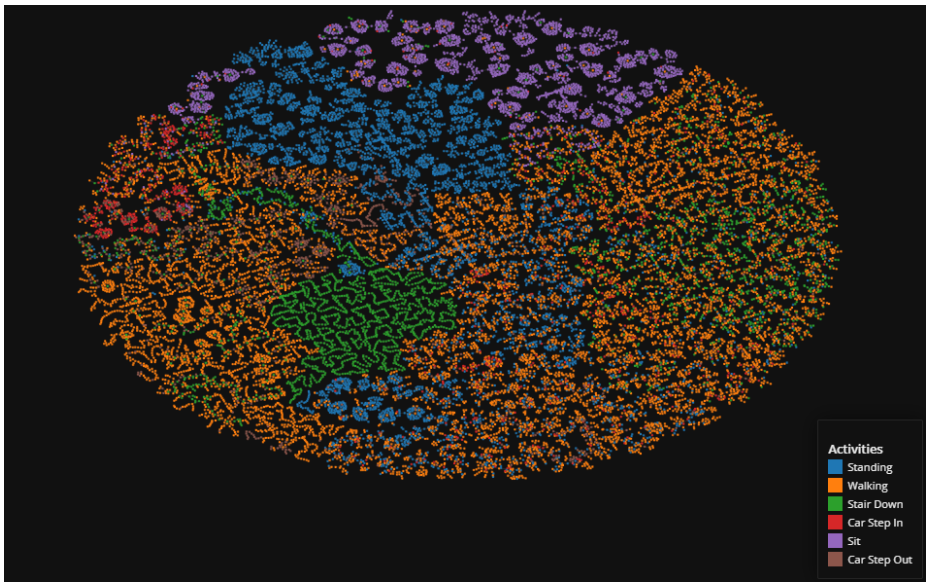

Figure 6.19: TMAP applied on the MobiAct SLH sub-dataset

while the STN action involves both vertical and horizontal movements. While we do not know exactly why the model learns this relationship between the two activities, the association shows up in many of the other plots of MobiAct data (see Figures 6.22 and 6.23), heavily implying that a hidden relationship does exist and requires deeper probing.

As with the other visualization algorithms, we apply TMAP onto the temporally-reduced HAR datasets to determine if any intuitive knowledge can be inferred from the cluster results. Figure 6.22 shows the full sequence MobiAct plotted using TMAP. Given the near-perfect performances for both classification and clustering, it is not surprising to see the visual clusters are clear and distinct, supporting the fact that supervised learning in combination with long temporal sequencing is capable of extracting excellent features. We also include the TMAP graphs for the raw MobiAct dataset as a comparison (Fig. 6.23). Immediately, we can understand how effective

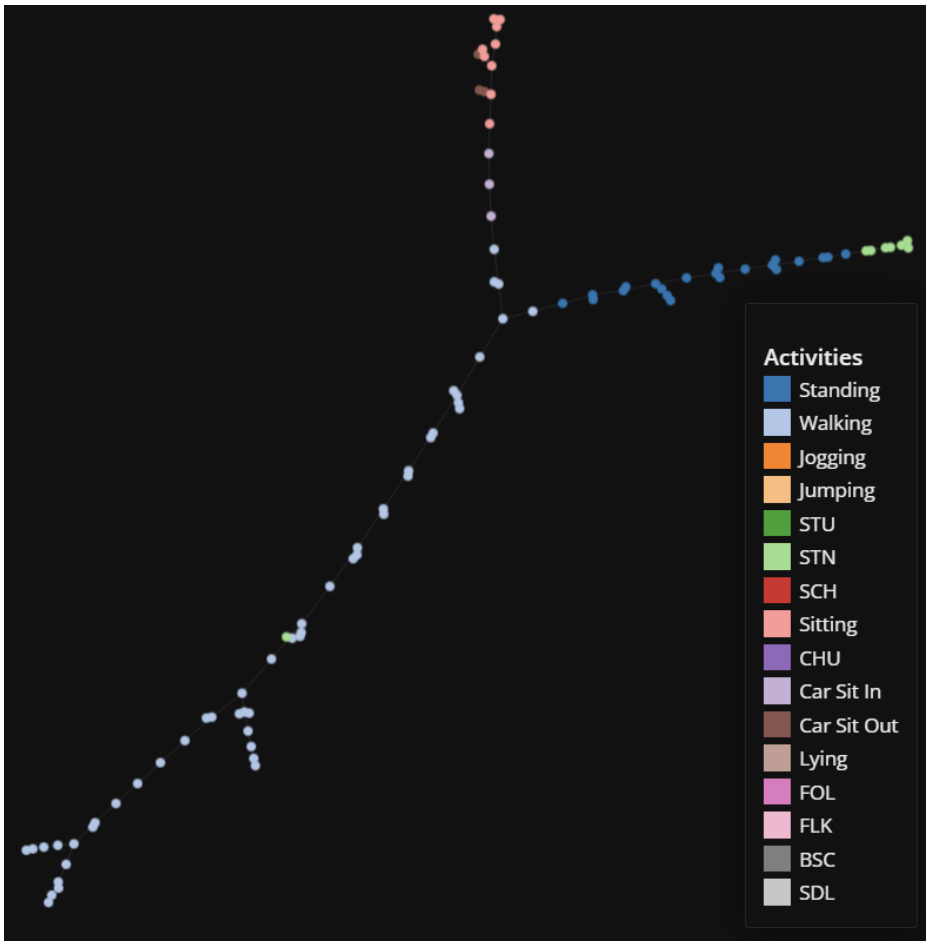

Figure 6.20: TMAP applied on the reduced MobiAct SLH sub-dataset

temporal feature extraction is when we compare the TMAP plots of the original MobiAct (Fig. 6.21) and the temporal-reduced MobiAct. Where the original MobiAct graph has the prominent actions (WAL and STD, etc.) dominate most of the space with the lesser activities weaved in-between, the temporally-extracted plots show distinct clusters between most of the activities. In the full sequence data, the classes are more distinct since the clustering performance is better overall while the raw data plot still provides clear relationships between the activities despite slightly-poorer cluster scores. Figure 6.23 highlights some of the intuition we can extract from visualizing

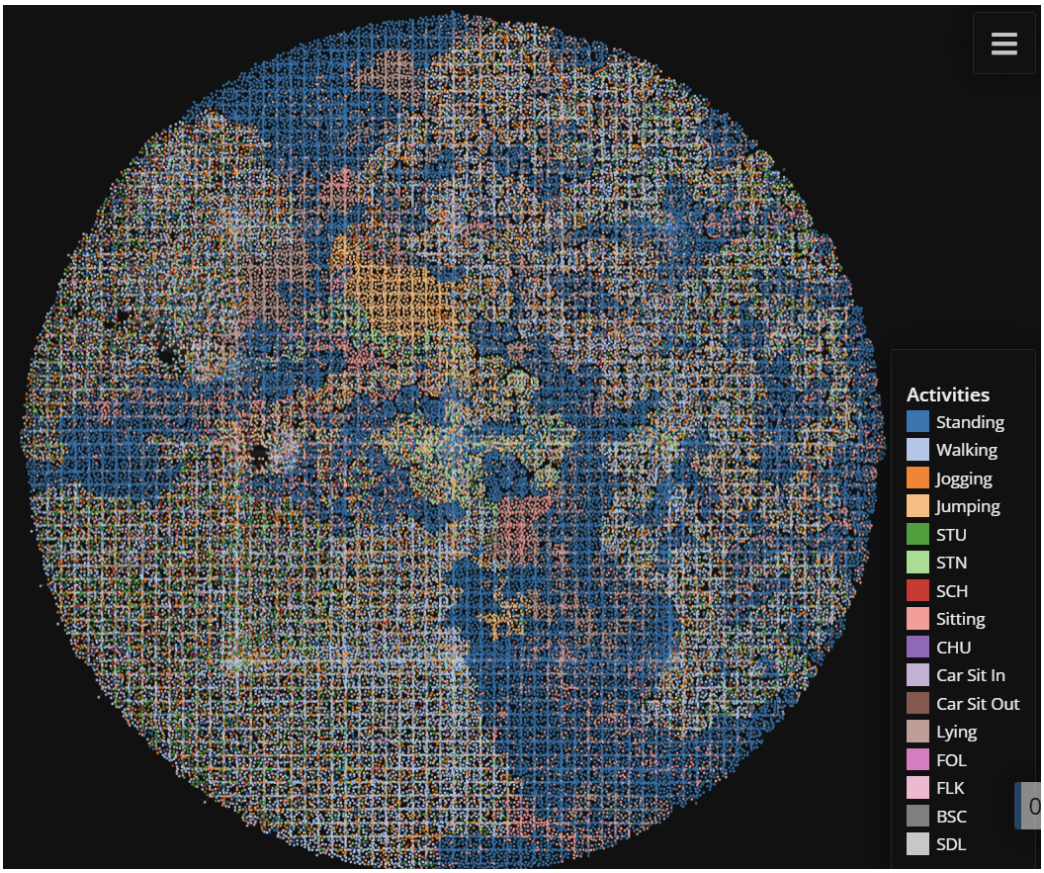

Figure 6.21: TMAP plot of the the entire MobiAct dataset

the the dataset. For example, we see understandable connections between activities such as WAL and JOG, and the triplets of CSI, SIT, and CSO. We can also observe some unintuitive connections like the STN-STD relationship we mentioned before, and undoubtedly, more relationships provided we dig deeper.

Additionally, since TMAP allows for clear and aesthetic visualization, we provide TMAP plots of the MobiAct dataset with varying window lengths. BY visualizing the clusters in this way, we are able to extrapolate a better reasoning for each corresponding cluster scores of the window sizes. For example, the plot of window size 512

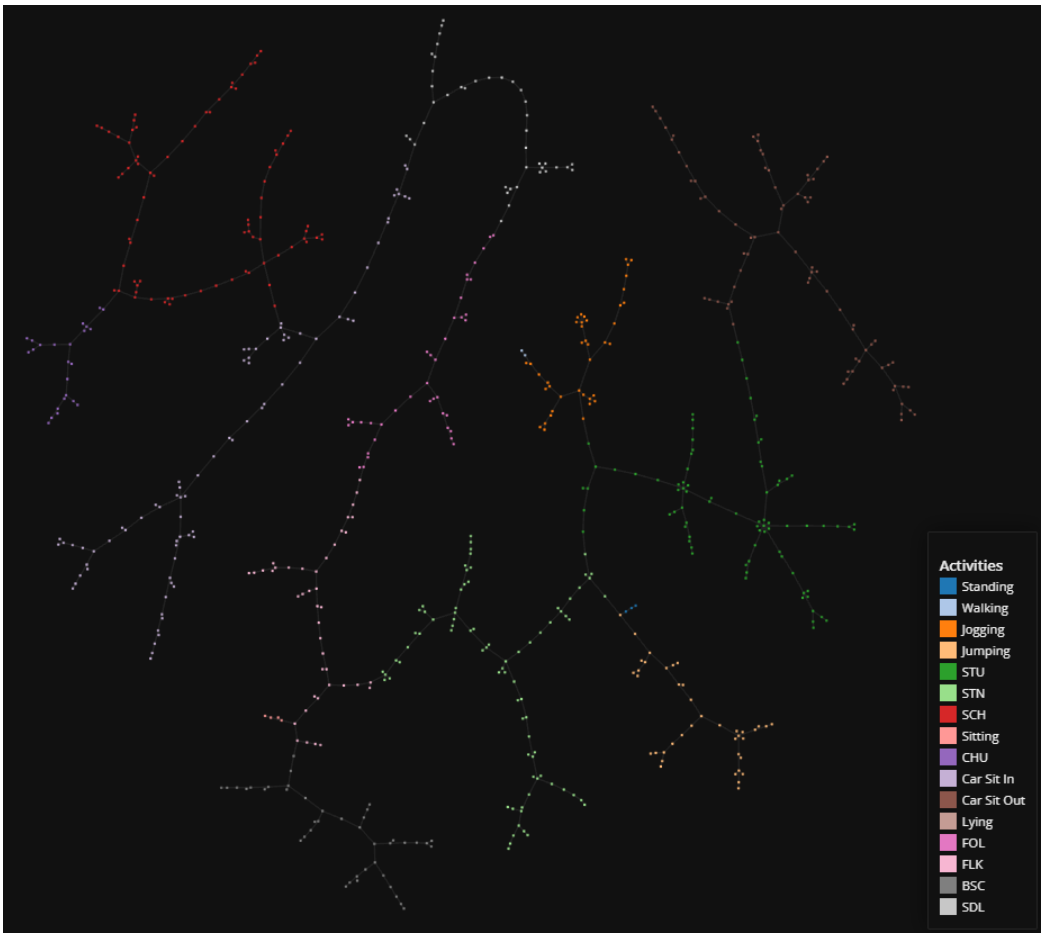

Figure 6.22: TMAP plot of the full sequence MobiAct dataset

6.24 presents a small but distinct cluster of SCH (red) points while the plot of window 1028 6.25 has noticeably much fewer points and are scattered around the graph. Similar patterns are observed for all the other lesser activities, which is a testament to the problem of the window bisecting sequences as mentioned in the previous chapter (Chapter 5). The plot for window size of 2048, while it returns excellent cluster scores, only features 5 activities, therefore we will disregard it.

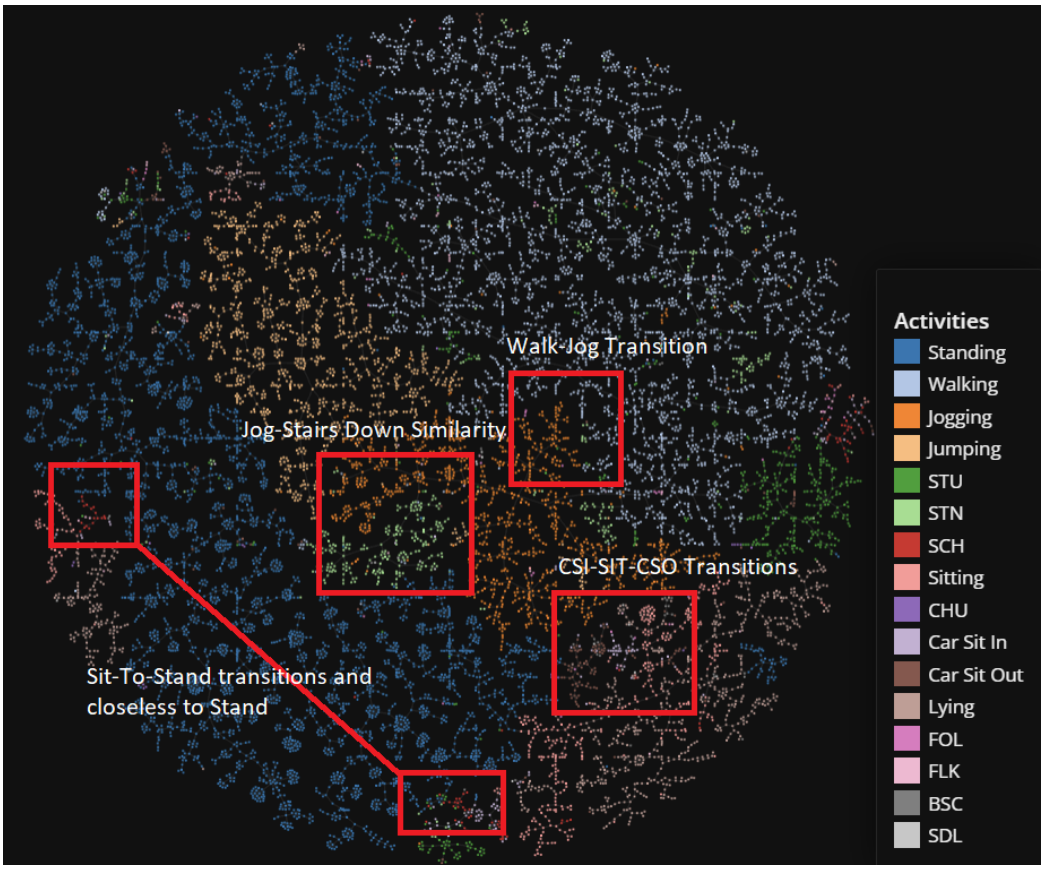

Figure 6.23: TMAP plot of the raw MobiAct dataset

ClusterPlot

ClusterPlot [86] is a 2021 algorithm that tries to stand out by preserving not only the data's pair-wise spatial relationships, but also the inter-cluster qualities, such as proximity and overlaps with other clusters. The author's priority is not to display the details of the data by plotting individual point (like TMAP does), but rather to present the topological distribution in a minimalistic fashion to convey the general characteristics within the embedded data. The resulting clusters are presented as minimalistic blobs to emphasize overlap and general spatial information.

The algorithm utilizes a form a k-nearest-neighbours to determine the membership

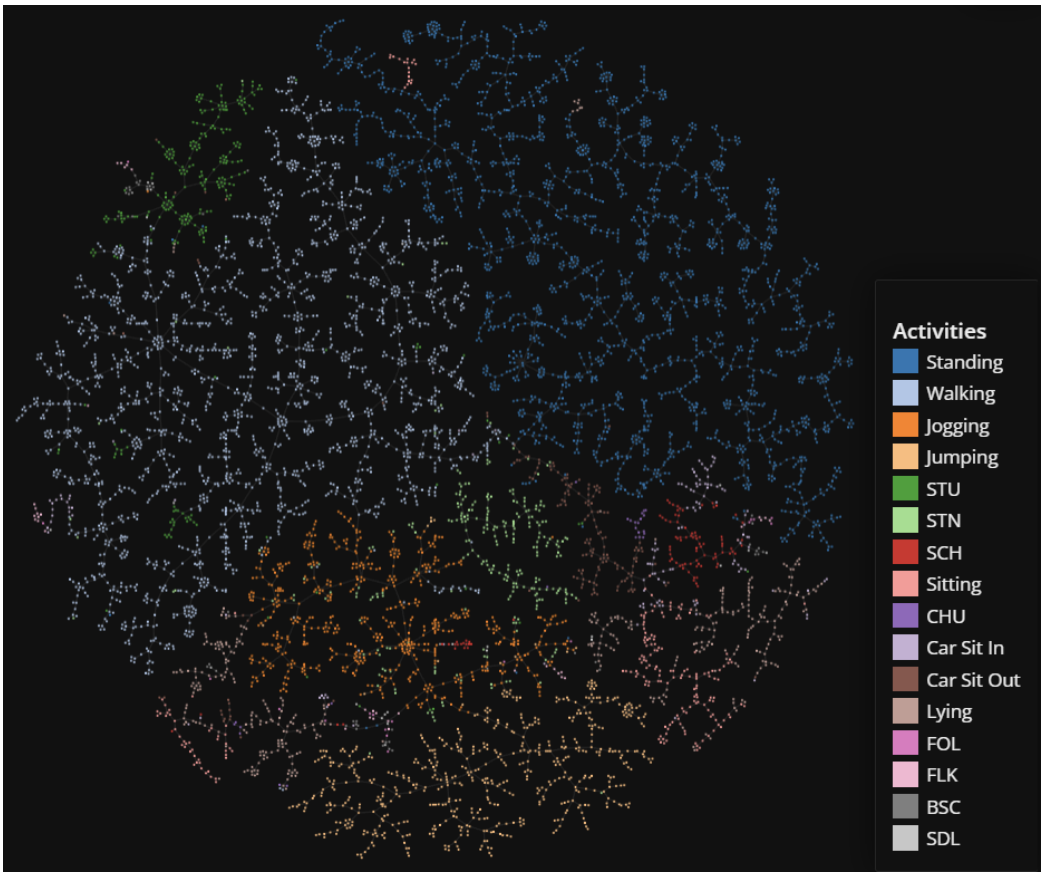

Figure 6.24: TMAP plot of the raw MobiAct dataset with window size of 512

of points within sub-clusters. Each larger cluster is made of numerous sub-clusters, and each sub-cluster is grounded around an anchor point. The algorithm uses these anchors to calculate the shape of the cluster. The anchors are initially projected onto a lower dimensional space and subsequently optimized to reduce the difference between inter-cluster relations in the low and high dimensional space. The algorithm uses UMAP as the dimensional reduction method and the BIRCH clustering algorithm to find intra-cluster information.

Fig 6.26 shows the ClusterPlot graph when applied to the dataset. As mentioned before, the algorithm tries to emphasize the general relationship between classes by

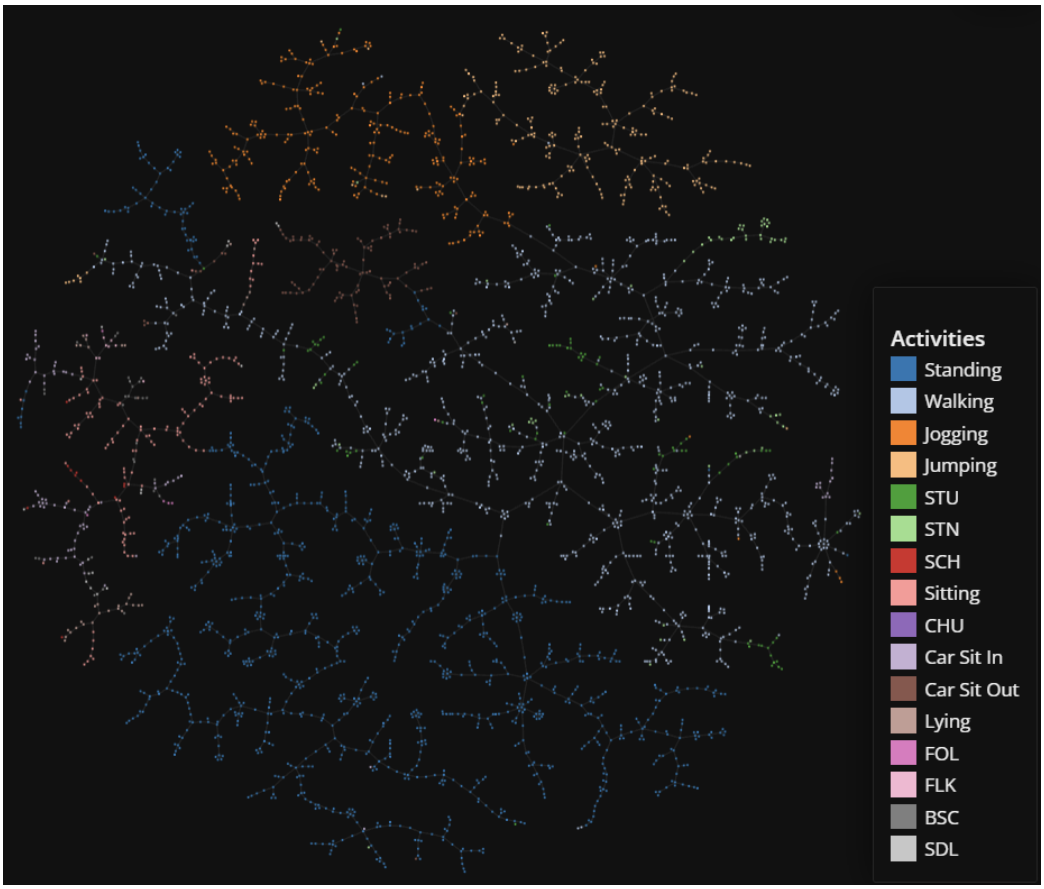

Figure 6.25: TMAP plot of the raw MobiAct dataset with window size of 1024

focusing on the overlaps of clusters. The results are as expected: the WAL class encapsulates a large majority of the other actions while some are distinct enough to be separated from the central blob. However, we can see that the quality of the separation is not as ideal compared to the other algorithms. The overlapping effect we observe in the plot is representative of the transitional nature we find in human activities. Since ClusterPlot creates the blob boundaries from the activity points at the extremities in the data space, overlapping blobs mean overlapping or intertwining data points. We can see the STN-STD relationship in this plot as well (at the top right), except contrary to what we observed in the TMAP plots, the STN points seem

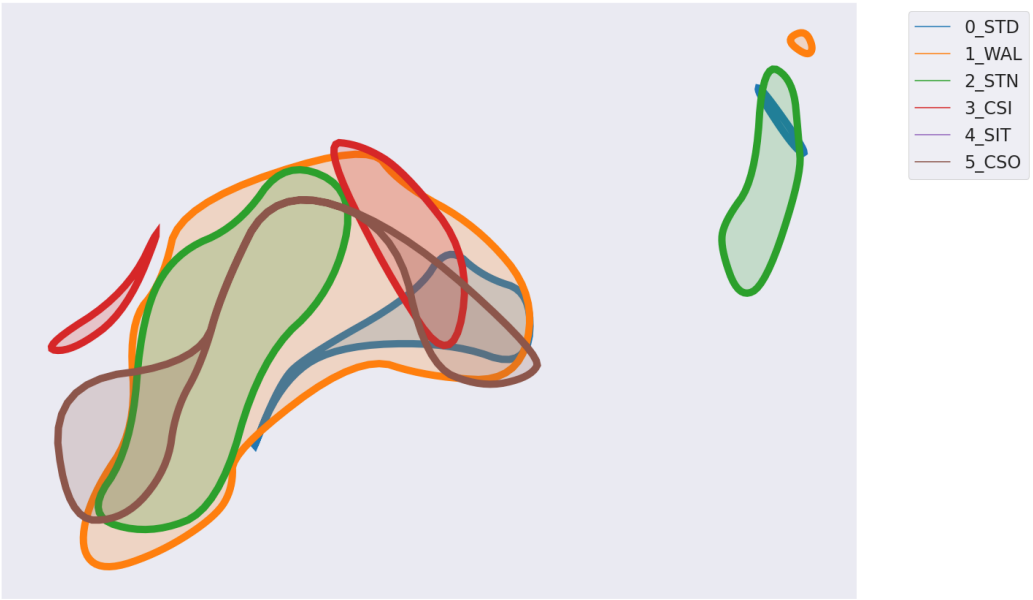

Figure 6.26: ClusterPlot output on the MobiAct SLH sub-dataset

to encapsulate the entirety of the STD points. From what we have seen before, the STD activities and the CSI-SIT-CSO sequence should be distinct enough to warrant distinct blobs, but from the image we can see this is not the case. We believe this might be due to inexperience with the algorithm parameters. ClusterPlot is a very difficult algorithm to use because the vast number (around 40) of parameters that can be configured. Additionally, testing the effects of the parameters values proved to be a challenge because training and graphing the data takes a long time relative to the other visualization algorithms. As a result, while ClusterPlot can return simplistic clusters for easy intuition, the cost might be too expensive considering the speed and detail provided by the other algorithms.

Below are the ClusterPlot graphs for the MobiAct raw dataset (Fig. 6.27) and the UCI dataset (Fig. 6.28). We can clearly see that the visualization quality is not as clear as the prior plots, but there is still merit in observing the general shapes of the clusters.

With the UCI ClusterPlot graph however, we can observe the distinct separation of clusters, which is expected given the high metrics the clustering algorithms return.

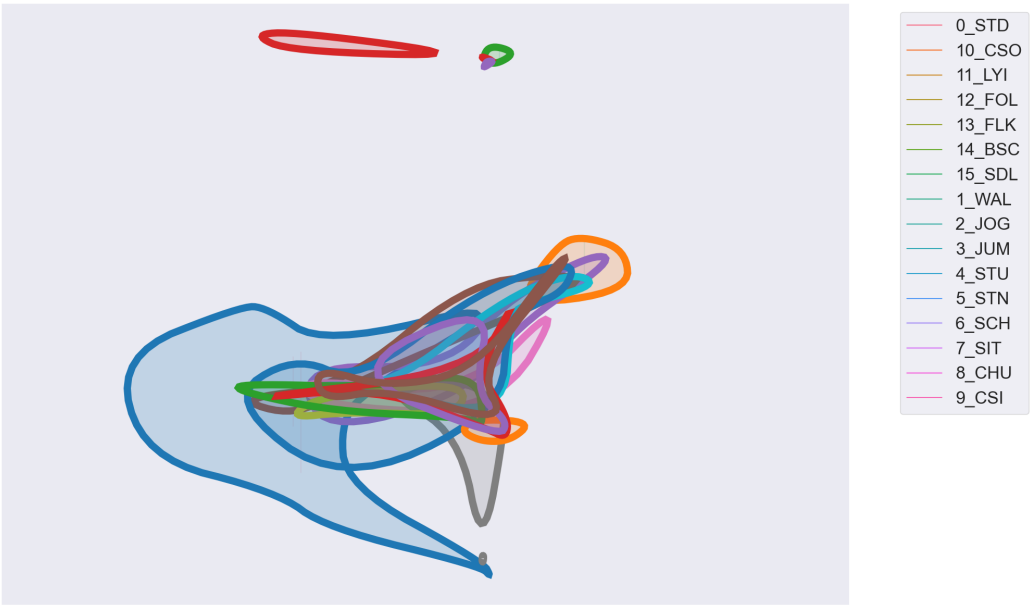

Figure 6.27: ClusterPlot output on the full MobiAct dataset

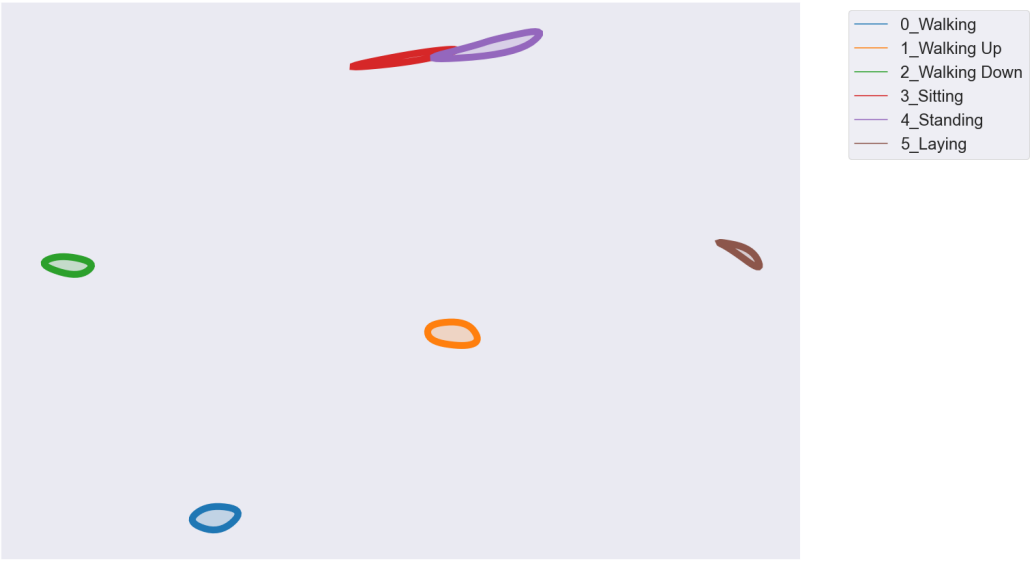

Figure 6.28: ClusterPlot output on the UCI dataset

6.5 Summary

This chapter presents several promising state-of-the-art dimensionality reduction algorithms and compares their visualizations to established algorithms. We applied the models towards high-dimensional complex human activity recognition sensor data and evaluated the resulting graphs using expert knowledge of the data. Interpretations of the plots were provided and compared to existing observable patterns in the dataset. The temporal-extracted features were fed into the modern visualization algorithms to provide a visual representation on the quality of temporal feature extraction models. From the plots returned by the visualization algorithms, we are able to extract both intuitive and unintuitive knowledge. We conclude that two of the modern algorithms (UMAP and TMAP) returns plots that improve the representations given by the foundational methods. The last model (ClusMap) produces simple overlapping clusters meant to emphasize the global structure of the data but might not be preferable given the high performance costs.

Chapter 7

Conclusion

7.1 Conclusion

In this thesis we have presented a journey that began with a single hypothesis and later evolved into an experimental odyssey through several conceptual disciplines with the specific goal of generating a deeper understanding of clustering temporal-based human activity recognition data streams. We began with a naive approach of relatively simple independent clustering models, and upon realizing the increased difficulty embedded within the nature of human movements, pivot our focus towards powerful machine learning models. The experimentation at this point determine that time-series IoT data cannot be effectively clustering to identify human activities without increasing computational power. Methodical analysis of proposed architectures from convolutional neural networks to different iterations of autoencoders reveal the necessity of temporal-based processing since temporal-based transitions between activities are difficult to identify and learn. This illuminates our next implementational steps. Integration of temporal analysis from long short-term memory models with the compressional abilities of convolutional neural networks allow for the creation of robust

time-dependent feature extraction models. With this architecture, temporal-based features are competently reduced, extracted, and ultimately fed into the simplistic clustering algorithms to return superior clusters. The addition of temporal processing allow deeper temporal analysis, ultimately returning greater insight into the evasive transitional actions of the dataset. Various visualizations are provided to raise deeper exploratory questions regarding the dataset and grant visual validation in a clear and aesthetically-pleasing manner. The conclusion of our exploration is clear: temporal-based feature-extraction supervised learning methods must be adopted alongside unsupervised clustering techniques to effectively return ideal temporal-based clusters from highly complex human activity recognition data streams.

In the first chapter, we introduce and highlight the main concepts and challenges present in live stream clustering of high-dimensional Human Activity Recognition (HAR) data. Mainly, any potential solutions must consider both the various problems in general stream processing and address the intricate and complex nature of human movements. Purely supervised learning methods are incapable of extracting subtleties or hidden features in the data, so eventually an unsupervised application must be introduced to uncover deeper concepts. From here, we present the pathway of our research into HAR feature extraction, beginning with an explanation of the concepts covered in the thesis, followed by an overview on the field of stream clustering and testing implementations of potential clustering algorithms, continuing with dimensional reduction AutoEncoder (AE) models, which are followed by powerful temporal-based feature extraction algorithms, and finally ending with aesthetic visualizations of the temporally-reduced features.

The second chapter presents the core concepts shared between almost all stream

clustering algorithms and explains the fundamental background information necessary for understanding the upcoming thesis content. The material covered in this chapter include windowing methods, type of processing stages, data summarization techniques, and general clustering approaches. A comprehensive timeline on the field of stream clustering was given from an evolutionary perspective, allowing readers to follow the advancements and understand how each iteration of algorithms builds upon the previous. Concise literature reviews on Autoencoders, LSTMs, and incremetal learning were also presented.

Chapter 3 includes our introductory experiments with relatively simplistic literature stream clustering algorithms. Preliminary tests with some of the literature models provided a basis to advance our research based on the observed results. Three algorithms were tested and evaluated on a specific sub-dataset from the MobiAct HAR collective. The first of the three algorithms is FISHDBC, a density-based algorithm derived from DBSCAN. The three algotihms were chosen because literature results show high performance across most datasets and the ability to create arbitrarily-shaped clusters is of great benefit for the sporadic and complex distribution of data in HAR data streams. FISHDBC was able to achieve 0.347 ARI and 0.551 NMI on the SLH sub-dataset. The second algorithm, WCDS, is a streaming implementation based on the classic WiSARD architecture, and is capable of memorizing binary representations of patterns to compare and match incoming input sequences. WCDS performed slightly better than FISHDBC, with an ARI score of 0.496 and NMI score of 0.567. DEC is the final algorithm we explored in this chapter, and the architecture consists of an AE model implemented with a custom clustering layer. DEC had the lowest performance of all the models, only returning score of 0.406 and 0.498, for ARI

and NMI respectively. The sub-optimal but relatively consistent scoring across three largely different architectures implies that some critical element is missing, therefore preventing the models from effectively clustering the HAR data. The most prevalent theory is that the algorithms are too simplistic in their design, and that a more complex and powerful architecture will allow for improved performance. This thought process leads us into the fourth chapter, where capable AE networks are utilized for both their data compression abilities and their increased computational prowess.

Chapter 4 presents our journey into exploring HAR feature extraction and reduction through powerful AE implementations, as standalone models and in combination with other architectures. We started with the previously implemented DEC model and retrained it on the full dataset, thinking that exposure to greater magnitudes of data can help improve model performance. However, the opposite occurs yet again, as the clustering scores drop significantly, achieving new lows of 0.32 NMI and 0.24 ARI for the MobiAct data and scores of 0.54 NMI and 0.49 ARI for the UCI data. With reference to only K-means performance for convenience, we first take a highly successful 2D CNN classification model to create a baseline for clustering quality with a NMI and ARI scores of 0.59 and 0.44, respectively. However, upon passing the convolutional features into the simple AE, we notice the scores slightly decrease as opposed to the expected increased that comes from feature reduction. With the lack of temporal information being the common denominator between the two experiments, we apply our first temporal-based model in the form of a LSTM AE, believing that while performance will mostly not be perfect, some improvement must be observed. The LSTM AE does perform better than the DEC model (0.49 NMI and 0.31 ARI for MobiAct; 0.51 NMI and 0.38 ARI for UCI), but still has yet to reach the baseline

established by the 2D CNN + AE combination. From here, a thought arose: perhaps even with temporal information, purely unsupervised methods are incapable of extracting good features for clustering. This led us into designing a hybrid architecture combining a convolutional AE with a temporal LSTM classifier to simultaneously compress and learn temporal dependencies within the sequences. This approach finally returns substantial improvements in both classification and clustering quality over the previous models, achieving mid 90's for accuracy and mid to high 80s for clustering. A subsequent experiment then explored the effect of training the hybrid model using a dataset without the fall activities and saw good improvements on the clustering metrics, implying that the fall activities add another layer of complexity that the model has trouble effectively processing.

In Chapter 5, temporal-extraction (TE) architectures are implemented to build on our previously established requirement for powerful neural network models in conjunction with our recently concluded need of temporal processing. We present a number of models, each a different iteration of the base TE model and each improving on its predecessor in regard to architecture complexity, layer variations, hyper-parameter deviations, or a combination of all three. Additionally, multiple variants of the MobiAct dataset are introduced and explored to determine the effects of dataset structure and pre-processing methods on the final performance. The first implementation has the simplest TE architecture and this initial experiment is meant to establish the validity of the approach and to set a baseline for performance. This model simply consists of a 1D convolution layer and a LSTM layer, and accounting for the final dense classification layers creates a hybrid model capable of extracting temporal-based features. This model achieved 92.5% classification accuracy on average across

both the MobiAct and UCI HAR datasets, showing that accounting for temporal dependencies within sequences results in good performances. The next implementation improves on the architecture by adding addition layers for increased computational power. Additionally, new dataset structures are introduced. This improved TE model improves MobiAct classification performance to 95% and UCI performance to a consistent 93%. The third time-distributed iteration re-formats the time sequences in a different manner, allowing the TE models to process multiple smaller segments in a sequence instead of analyzing full sequences, thus improving computational speeds. Depending on the dataset, this model architecture achieves either from 92 and 93.5% to a near-perfect 98%. Clustering on the temporally-reduced features also begins from this iteration, with the initial standard for clustering performance established as an average of 0.64 ARI/NMI using K-means and 0.44 ARI/NMI using FISHDBC on two MobiAct datasets and 0.92 ARI/NMI using K-means and 0.86 ARI/NMI using FISHDBC on the UCI dataset. It is notable that the full sequence MobiAct dataset achieves near-perfect ARI and NMI scores at 0.99 each. The final iteration architecture once again improves on computational power by replacing the singular CNN head of the the time-distributed model with three CNN heads, with the motivation to extract temporal knowledge at various levels of abstractions. In exchange for increased computational resources, this iterations gains a slight increase in classification for all datasets (+ 0.5 percent points), but the largest improvement is in the clustering performance. The tri-head model raises NMI with K-means to around 0.76 and 0.65 with FISHDBC for the MobiAct data. The UCI ARI/NMI scores also reaches 0.99, matching the performance for the full sequence MobiAct data. To determine the affect of window length on clustering performance, an experiment was carried out

using the time-distributed model with varying window sizes. We established while the full sequences will return excellent performance, expert knowledge is necessary to determine the best window length, since too small of a value will prevent the model from learning overarching patterns while too large of a length will remove smaller activities from being recognized.

Finally, the last chapter focuses on visualization of high-dimensional clusters for manual validation and intuitive visual understanding of results. A short review of modern visualization literature algorithms is given to provide some context into the field. Two fundamental dimensionality reduction (and typically followed by visualization) algorithms, t-SNE and PCA, are explained to set the foundation and baselines of the chapter. After, multiple modern visualization algorithms are presented and applied onto the MobiAct sub-dataset used in Chapter 3 in addition to the temporally-reduced data from Chapter 6. The visual graphs allow us to derive intuitive knowledge about the dataset and evaluate the quality of the temporal-dependent clusters.

7.2 Future Work

The direction of future work is for deeper analysis of HAR clusters to undercover hidden knowledge pertaining to the hierarchy of human actions. Since unsupervised learning models return results created from their own conceptual definitions, the unintuitive representation of the data holds potential for unpredictable knowledge, and dissecting the nature of these clusters can allow for advanced study. The work with TE and the temporally-reduced clusters are the precursor for unlocking deeper understanding of human movements. We can adjust parameters for the clustering

algorithms to manipulate the number and distance of clusters to determine the relationship between particular activities. For example, if increasing the number of clusters returns an extra group between the *walking* and *jogging* actions or as an overlap between the two, then there is a possibility the algorithm is telling us that *fast walking* is a distinct activity. Specifically, future work involves advanced analysis of the temporally-extracted HAR features to construct a hierarchical understanding on the nature of human activities.

References

[1] C.C. Aggarwal, S.Y. Phillip, J. Han, and J. Wang. A framework for clustering evolving data streams. In *Proceedings of the 2003 VLDB conference*, pages 81–92, Berlin, Germany, 9-12 September 2003.

[2] Antonio A. Aguileta, Ramon F. Brena, Oscar Mayora, Erik Molino-Minero-Re, and Luis A. Trejo. Multi-sensor fusion for activity recognition - a survey. *MDPI Sensors*, 19(17):3808, 2019.

[3] D. Ajerla, S. Mahfuz, and F. Zulkernine. Fall detection from physical activity monitoring data. In *Proceedings of the 7th International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 1–9, London, England, United Kingdom, 20 August 2018.

[4] Dharmitha Ajerla, Sazia Mahfuz, and Farhana Zulkernine. A real-time patient monitoring framework for fall detection. *Wirel. Commun. Mob. Com.*, 2019:1–13, 2019.

[5] I. Aleksander, W. Thomas, and P. Bowden. Wisard: A radical step forward in image recognition. *Sensor Rev.*, 4:1–4, 1984.

[6] Bandar Almaslukh, Jalal AlMuhtadi, and Abdelmonim Artoli. An effective deep autoencoder approach for online smartphone-based human activity recognition. *IJCSNS international journal of computer science and network security*, 17(4):160–164, 2017.

[7] D. Anguita, A. Ghio, L. Oneta, X. Parra, and J.L. Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *Proceedings of the 21st European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 1–3, Bruges, Belgium, 24-26 April 2013.

[8] Mohammed Oualid Attaoui, Hanene Azzag, Mustapha Lebbah, and Nabil Keskes. Improved multi-objective data stream clustering with time and memory optimization, 2022.

[9] H. Azzag, N. Monmarche, M. Slimane, and G. Venturini. Anttree: A new model for clustering with artificial ants. In *Proceedsings of the 2003 Congress on Evolutionary Computation, 2003. CEC '03.*, volume 4, pages 2642–2647, Canberra, Australia, 8-12 December 2003.

[10] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, volume 27 of *Proceedings of Machine Learning Research*, pages 37–49, Bellevue, Washington, USA, 02 Jul 2012. PMLR.

[11] A. Baldominos, Y. Saez, and P.A. Isasi. Survey of handwritten character recognition with mnist and emnist. *Appl. Sci.*, 9, 2019.

[12] JP. Barddal, H.M. Gomes, and F. Enembreck. Sncstream: A social network-based data stream clustering algorithm. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 935–340, Salamanca, Spain, 13-17 April 2015.

[13] JP. Barddal, H.M. Gomes, F. Enembreck, and JP. Barth. Sncstream+: Extending a high quality true anytime data stream clustering algorithm. *Inform. Syst.*, 62:60–73, 2016.

[14] Edgar A. Bernal, Xitong Yang, Qun Li, Jayant Kumar, Sriganesh Madhvanath, Palghat Ramesh, and Raja Bala. Deep temporal multimodal fusion for medical procedure monitoring using wearable sensors. *IEEE Transactions on Multimedia*, 20(1):107–118, 2018.

[15] E. Bertini, A. Tatu, and D. Keim. Quality metrics in high-dimensional data visualization: An overview and systematization. *IEEE Transactions on Visualization and Computer Graphics*, 17:2203–2212, 2011.

[16] B. Bhat, N. Tran, H. Shill, and U.Y. Ogras. W-har: An activity recognition dataset and framework using low-power wearable devices. *Sensors*, 20, 2020.

[17] Michele Borassi, Alessandro Epasto, Silvio Lattanzi, Sergei Vassilvitskii, and Morteza Zadimoghaddam. Sliding window algorithms for k-clustering problems. In *Proceedings of advances in neural information processing systems 33*, 2020.

[18] MR. Bouguelia, Y. Belaïd, and A. Belaïd. An adaptive incremental clustering method based on the growing neural gas algorithm. In *Proceedings of*

ICPRAM: the 2nd International Conference on Pattern Recognition Applications and Methods, pages 42–49, Barcelona, Spain, 15-18 February 2013.

[19] R.J.G.B. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In *Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 160–172, Gold Coast, Australia, 14-17 April 2013.

[20] F. Cao, M. Ester, W. Qian, and A. Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM International Conference on Data Mining*, pages 328–339, Bethesda, Maryland, USA, 20-22 April 2006.

[21] D.O. Cardoso, F. MG. França, and J. Gama. Wcds: A two-phase weightless neural system for data stream clustering. *New Generat. Comput.*, 35:391–416, 2017.

[22] S.K. Challa, A. Kumar, and V.B. Semwal. A multibranch cnn-bilstm model for human activity recognition using wearable sensor data. *Visual Computing*, 2021.

[23] L. Chambers, M.M. Gaber, and Z.S. Abdallah. Deepstreamce: A streaming approach to concept evolution detection in deep neural networks, 2020.

[24] DM. Chan, R. Rao, F. Huang, and J.F. Canny. T-sne-cuda: Gpu-accelerated t-sne and its applications to modern data. In *Proceedings of 30th International Symposium on Computer Architecture and High Performance Computing*, pages 330–338, 2018.

[25] C. Chatzaki, M. Pediaditis, G. Vavoulas, and M. Tsiknakis. Human daily activity and fall recognition using a smartphone's acceleration sensor. In *Proceedings of the International Conference on Information and Communication Technologies for Ageing Well and E-health*, pages 100–118, April Rome, Italy, 21-22 April 2016.

[26] R. Chavarriage, H. Sagha, A. Calatroni, S. T. Digumarti, G. Troster, J. del R. Millan, and D. Roggen. The opportunity challenge: A benchmark database for on-body sensor-based activity recognition. *Pattern Recogn. Lett.*, 34:2033–2042, 2013.

[27] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, jun 2002.

[28] H. Chen, S. Mahfuz, and F. Zulkernine. Smart phone based human activity recognition. In *Proceedings of the 2019 IEEE International Conference on Bioinformatics and Biomedicine*, pages 2525–2532, San Diego, California, USA, 18-21 November 2019.

[29] Xin Cheng, Lei Zhang, Yin Tang, Yue Liu, Hao Wu, and Jun He. Real-time human activity recognition using conditionally parametrized convolutions on mobile and wearable devices. *IEEE Sensors Journal*, pages 1–1, 2022.

[30] Y. Cui, S. Surpur, S. Ahmad, and J. Hawkins. A comparative study of htm and other neural network models for online sequence learning with streaming data. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1530–1538, Vancouver, BC, Canada, 24-29 July 2016.

[31] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the 14th annual ACM symposium on Theory of computing*, pages 537–546, 2008.

[32] M. Dell'Amico. Fishdbc: Flexible and incremental, scalable hierarchical density-based clustering for arbitrary data and distance, 2019.

[33] Salah Ud Din, Junming Shao, Jay Kumar, Waqar Ali, Jiaming Liu, and Yu Ye. Online reliable semi-supervised learning on evolving data streams. *Information Sciences*, 525:152–171, 2020.

[34] J. Ding, A. Condon, and S.P. Shah. Interpretable dimensionality reduction of single-cell trasncriptome data with deep generative models. *Nature Communications*, 9, 2018.

[35] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbour graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, pages 577–586, 2011.

[36] G. Ellis and A. Dix. A taxonomy of clutter reduction for information visualisation. *IEEE Transactions on Visualization and Computer Graphics*, 13:1216–1223, 2007.

[37] M. Ester, HP. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 226–231, Portland, Oregon, USA, 2-4 August 1996.

[38] A. Ferarri, D. Micucci, M. Mobilio, and P. Napoletano. Deep learning and model personalization in sensor-based human activity recognition. *Journal of Reliable Intelligent Environments*, 2022.

[39] B. Fritzke. A self-organizing network that can follow non-stationary distributions. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 613–618, October Lausanne, Switzerland, 8-10 October 1997.

[40] S. Furao and O. Hasegawa. An incremental network for on-line unsupervised classification and topology learning. *Neural Networks*, 19:90–106, 2006.

[41] S. Furao, T. Ogura, and O. Hasegawa. An enhanced self-organizing incremental neural network for online unsupervised learning. *Neural Networks*, 20:893–903, 2007.

[42] Xile Gao, Haiyong Luo, Qu Wang, Fang Zhao, Langlang Ye, and Yuexia Zhang. a human activity recognition algorithm based on stacking denoising autoencoder and lighgbm. *sensors*, 19(4):947, 2019.

[43] D. Garcia-Gonzalez, D. Rivero, E. Fernandez-Blanco, and M.R. Luaces. A public domain dataset for real-life human activity recognition using smartphone sensors. *Sensors*, 20:2200, 2020.

[44] Alexander Gepperth and Barbara Hammer. Incremental learning algorithms and applications. In *proceedings of the European symposium on artificial neural networks*, 2016.

[45] M. Ghesmoune, M. Lebbah, and H. Azzag. A new growing neural gas for clustering data streams. *Neural Networks*, 78:36–50, 2016.

[46] M. Ghesmoune, M. Lebbah, and H. Azzah. State-of-the-art on clustering data streams. *Big Data Analytics*, 2016.

[47] Sujan Ghimire, Ravinesh C. Deo, Hua Wang, Mohanad S. Al-Musaylh, David Casillas-Pèrez, and Sancho Salcedo-Sanz. Stacked lstm sequence-to-sequence autoencoder with feature selection for daily solar radiation prediction: A review and new modeling results. *energies*, 15(3):1061, 2022.

[48] Google. Static vs. dynamic training. `https://developers.google.com/machine-learning/crash-course/static-vs-dynamic-training/video-lecture`. Accessed: 2021-11-6.

[49] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[50] M. Hassani, P. Spaus, A. Cuzzocrea, and T. Seidl. I-hastream: Density-based hierarchical clustering of big data streams and its application to big graph analytics tools. In *Proceedings of the 16th IEEE/ACM International Symposium on Cluster*, pages 656–665, Cartagena, Columbia, 16-19 May 2016.

[51] M. Hassani, P. Spaus, and T. Seidl. Adaptive multiple-resolution stream clustering. In *Proceedings of the International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pages 134–148, St. Petersburg, Russia, 21-24 July 2014.

[52] J. Heinrich and D. Weiskopf. State of the art of parallel coordinates. *Springer Eurographic*, pages 95–116, 2013.

[53] Sepp Hochreither. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(2):107–116, 1998.

[54] Sepp Hochreither and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[55] X. Hou, L. Shen, K. Sun, and G. Qiu. Deep feature consistent variational autoencoder. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision*, pages 1133–1141, Santa Rosa, CA, USA, 24-31 March 2017.

[56] Ruohe Huang, Ruliang Xiao, Weifu Zhu, Ping Gong, Jinhui Chen, and Imad Rida. towards an efficient real-time kernel function stream clustering method via shared nearest-neighbour density for iiot. *information sciences*, 566:364–378, 2021.

[57] Richard Hyde, Plamen Angelov, and A.R. MacKenzie. Fully online clustering of evolving data streams into arbitrarily shaped clusters. *information sciences*, 382–383:96–114, 2017.

[58] Md. Kamrul Islam, Md. Manjur Ahmed, and Kamal Z. Zamli. a buffer-based online clustering for evolving data stream. *information sciences*, 489:113–135, 2019.

[59] J. Jakubowski, P. Stanisz, S. Bobek, and G.J. Nalepa. Anomaly detection in asset degradation process using variational autoencoder and explanations. *Sensors*, 22:291, 2021.

[60] Ahmad Jalal, Shaharyar Kamal, and Daijin Kim. A depth video sensor-based life-logging human activity recognition system for elderly care in smart indoor environments. *Sensors*, 14(7):11735–11759, 2014.

[61] C. Jobanputra, J. Bavishi, and N. Doshi. Human activity recognition: A survey. *Procedia Comput. Sci.*, 155:698–703, 2019.

[62] J. Kerher and H. Helwig. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 19:495–513, 2012.

[63] I. Khan, J.Z. Huang, and K. Ivanov. Incremental density-based ensemble clustering over evolving data streams. *Neurocomputing*, 191:34–43, 2016.

[64] Imran Ullah Khan, Sitara Afzal, and Jong Weon Lee. Human activity recognition via hybrid deep learning based model. *Sensors*, 22(1), 2022.

[65] Zafar A. Khan and Won Sohn. Abnormal human activity recognition system based on r-transform and kernel discriminant technique for elderly home care. *IEEE Transactions on Consumer Electronics*, 57(4):1843–1850, 2011.

[66] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biol. Cybern.*, 43:59–69, 1982.

[67] M. Kontaki, A. Gounaris, A.N. Papadopoulos, K. Tsichlas, and Y. Manolopuolos. Efficient and flexible algorithms for montoring distance-based outliers over data streams. *Inform. Syst.*, 55:37–53, 2016.

[68] P. Krajsic and B. Franczyk. Variational autoencoder for anomaly detection in event data in online process mining. In *Proceedings of the 23rd International Conference on Enterprise Information Systems*, pages 567–574, Online Streaming, April 26-28, 2021.

[69] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The clustree: Indexing micro-clusters for anytime stream mining. *Knowl. Inf. Syst.*, 29:249–272, 2011.

[70] H. Kremer, P. Kranen, T. Jansen, T. Seidl, A. Bifet, G. Holmes, and B. Pfahringer. An effective evaluation measure for clustering on evolving data streams. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 868–876, San Diego, California, USA, 21-24 August 2011.

[71] S. Kullback and R.A. Leibler. On information and sufficiency. *Ann. Math Stat.*, 22:79–86, 1952.

[72] J.R. Kwapisz, G.M. Weiss, and S.A. Moore. Activity recognition using cell phone accelerometers. *ACM SIGKDD*, 12:1–8, 2010.

[73] Sirisup Laohakiat and Vera Sa-ing. An incremental density-based clustering framework using fuzzy local clustering. *Inform. Sciences*, 547:404–426, 2021.

[74] Yanchao Li, Yongli Wang, Qi Liu, Cheng Bi, Xiaohui Jiang, and Shurong Sun. Incremental semi-supervised learning on streaming data. *Pattern Recogn.*, 88:383–396, 2019.

[75] Weiyao Lin, Ming-Ting Sun, Radha Poovandran, and Zhengyou Zhang. Human activity recognition for video surveillance. In *2008 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 2737–2740, 2008.

[76] G.C. Linderman, M. Rachh, J.G. Hoskins, S. Steinerberger, and Y. Kluger. Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature Methods*, 16:243–245, 2019.

[77] Xiao Ling and Daniel S. Weld. Temporal information extraction. In *proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[78] Hao Liu and Xiao-juan Ban. Clustering by growing incremental self-organizing neural network. *expert systems with applications*, 42(11):4965–4981, 2015.

[79] L. Liu, S. Wang, B. Hu, Q. Qiong, J. Wen, and D.S. Rosenblum. Learning structures of interval-based bayesian networks in probabilistic generative model for human complex activity recognition. *Pattern Recogn.*, 81:545–561, 2018.

[80] S. Liu, D. Maljovec, B. Wang, PT. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE Transactions on Visualization and Computer Graphics*, 23:1249–1268, 2017.

[81] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics training: multi-class incremental learning without forgetting. In *2020 conference on computer vision and pattern recognition*, page 2002.10211, 2020.

[82] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret. Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot. *Sensors*, 17, 2017.

[83] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, Oakland, California, USA, 18-21 June 1965 and December 27 1965-January 7 1966.

[84] Sazia Mahfuz, Haruna Isah, Farhana Zulkernine, and Pete Nicholls. Detecting irregular patterns in iot streaming data for fall detection. In *Proceedings of the IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, pages 588–594, Vancouver, British Columbia, Canada, 1-3 November 2018.

[85] Jose Maia, Carlos Alberto Severiano Junior, Guimar aes, Cristiano Leite de Castro, André Paim Lemos, Juan Camilo Fonseca Galindo, and Miri Weiss Cohen. evolving clustering algorithm based on mixture of typicalities for stream data mining. *Future generation computer systems*, 106:672–684, 2020.

[86] O. Malkai, M. Lu, and D. Cohen-Or. Clusterplot: High-dimensional cluster visualization. *arXiv*, page 2103.02992, 2021.

[87] Y.A. Malkov and D.A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE T. Pattern. Anal.*, 42:824–836, 2020.

[88] S. Mansalis, E. Ntoutsi, N. Pelekis, and Y. Theodoridis. An evaluation of data stream clustering algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, pages 167–187, 2018.

[89] T.M. Martinez, S.G. Berkovich, and K.J. Schulten. 'neural-gas'- network for vector quantization and its application to time-series prediction. *IEEE T. Neural Networ.*, 4:558–569, 1993.

[90] L. McIness, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv*, 1802.03426, 2018.

[91] Yong ming Wu, Lin sheng Chen, Shao bo Li, and Jia dui Chen. An adaptive algorithm for dealing with data stream evolution and singularity. *Inform. Sciences*, 545:312–330, 2021.

[92] K.R. Moon, D. van Dijk, Z. Wang, and et al. Visualizing structure and transitions in high-dimensional biological data. *Nature Biotechnology*, 37:1482–1492, 2019.

[93] M. Mousavi, A. A. Bakar, and M. Vakilian. Data stream clustering algorithms: A review. *Int. J. Adv. Soft Comput. Appl*, 7:1–15, 2015.

[94] Sebastian Münzer, Philip Schmidt, Attila Reiss, Michael Hanselmann, Rainer Stiefelhagen, and Robert Dürichen. cnn-based sensor fusion techniques for multimodal human activity recognition. In *ISWC'17:Proceedings of the 2017 ACM international symposium on wearable computers*, ACM, pages 158–165, 2017.

[95] A. Narayan, B. Berger, and H. Cho. Density-preserving data visualization unveils dynamic patterns of single-cell transcriptomic variability. *Nature Biotechonology*, 39:765–774, 2020.

[96] Tien Thanh Nguyen, Manh Truong Dang, Anh Vu Luong, Alan Wee-Chung Liew, Tiancai Liang, and John McCall. Multi-label classification via incremental clustering on an evolving data stream. *Pattern Recogn.*, 95:96–113, 2019.

[97] Henry Friday Nweke, Ying Wah Teh, Uzoma Rita Alo, and Ghulam Mujtaba. Analysis of multi-sensor fusion for mobile and wearable sensor based human activity recognition. In *ICDPA 2018: Proceedings of the International Conference on Data Processing and Applications*, ICDPA, pages 22–26. ACM, 2018.

[98] Henry Friday Nweke, Teh Ying Wah, Ghulam Mujtaba, and Mohammed Ali Al-garadi. Data fusion and multiple classifier systems for human activity detection and health monitoring: Review and open research directions. *Information Fusion*, 46:147–170, 2019.

[99] Godwin Ogbuabor and Robert La. human activity recognition for healthcare using smartphones. In *ICMLC 2018: Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, pages 41–46, 2018.

[100] Banos Oresit, Juan-Manuel Galvez, Miguel Damas, Hector Pomares, and Ignacio Rojas. Window size impact in human activity recognition. *Sensors*, 14(4):6474–6499, 2014.

[101] R. Pari, M. Sandhya, and Sharmila Sankar. a multi-tier stacked ensemble algorithm to reduce the regret of incremental learning for streaming data. *IEEE Access*, 6:48726–48739, 2018.

[102] Karl F.R.S. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[103] David Pratella, Samira Ait-El-Mkadem Saadi, and Sylvie Bannwarth. A survey of autoencoder algorithms to pave the diagnosis of rare diseases. *international journal of molecular sciences*, 22(19):10891, 2021.

[104] D. Probst and JL. Reymond. Visualization of very large high-dimensional data sets as minimum spanning trees. *Cheminformatics*, 12, 2020.

[105] Daniel Puschmann, Payam Barnaghi, and Rahim Tafazolli. Adaptive clustering for dynamic iot data streams. *IEEE internet of things journal*, 4(1):64–74, 2016.

[106] Sergio Ramírez-Gallego, Bartosz Krawczyk, Salvador García, MichałWoźniak, and Francisco Herrera. A survey on data preprocessing for data stream mining: Current status and future directions. *Neurocomputing*, 239:39–57, 2017.

[107] Gheorghe Sebestyen, Ionut Stoica, and Anca Hangan. Human activity recognition and monitoring for elderly people. In *2016 IEEE 12th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 341–347, 2016.

[108] M. Shoaib, S. Bosch, O.D. Incel, H. Scholten, and P.J. Havinga. A survey of online activity recognition using mobile phones. *Sensors*, 15:2059–2085, 2015.

[109] Muhammad Shoaib, Stephan Bosch, Ozlem D. Incel, Hans Scholten, and Paul J.M. Havinga. Complex human activity recognition using smartphone and wrist-worn motion sensors. *Sensors*, 16(4):426, 2016.

[110] N. Sikder and A. Ku-har Nahib. An open dataset for heterogeneous human activity recognition. *Pattern Recogn. Lett*, 146:46–54, 2021.

[111] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschkam, and Joao de Carvalho Andre. C.P.L.F.and Gama. Data stream clustering: A survey. *ACM Computing Survey*, 2013.

[112] V. Svensson, R. Vento-Tormo, and S.A. Teichmann. Exponential scaling of single-cell rna-seq in the past decade. *Nature Protocols*, 13:599–604, 2018.

[113] Ahmed Taha, Hala H. Zayed, M.E. Khalifa, and El-Sayed M. El-Horbaty. Human activity recognition for surveillance applications. In *ICIT 2015 The 7th International Conference on Information Technology*, 2015.

[114] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proceedings of the 25th International Conference on World Wide Web*, pages 287–297, Montreal, Quebec, Canada, April 11-15 2016.

[115] M. Tavallaee, E. Bagheri, W. Lu, and A.A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Security and Defence Applications*, pages 1–6. IEEE, Ottawa, Ontario, Canada, 8-10 July 2009.

[116] Dipanwita Thakur, Suparna Biswas, Edmond S. L. Ho, and Samiran Chattopadhyay. Convae-lstm: Convolutional autoencoder long short-term memory network for smartphone-based human activity recognition. *IEEE Access*, 10:4137–4156, 2022.

[117] Mary E. Tinetti and Christianna S. Williams. Falls, injuries due to falls, and the risk of admission to nursing home. *England journal of medicine*, 337(18):1279–1284, 1997.

[118] Gamze Uslu, Sebnem Baydere, Ata Tekin, and Feryal Subaşı. Accelerometer based classification of elbow flexion and extension exercises. In *2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2657–2663, 2020.

[119] L. van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245, 2014.

[120] L. van der Maater and G.E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[121] Alireza Abdein Varamin, Ehsan Abbasnejad, Qinfeng Shi, Damith C. Ranasinghe, and Hamid Rezatofighi. Deep auto-set: a deep auto-encode-set network for activity recognition using wearables. In *MobiQuitous'18: Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, MobiQuitous, pages 246–253, 2018.

[122] Michalis Vrigkas, , Nikou Christophoros, and Kakadiaris A. Ioannis. A review of human activity recognition methods. *Frontiers in Robotics and AI*, 2, 2015.

[123] Y. Wang, K. Yang, X. Jing, and H.L. Jin. Problems of kdd cup 99 dataset existed and data preprocessing. *Appl. Mech. Mater.*, 667:218–225, 2014.

[124] Yan Wang, Shuang Cang, and Hongnian Yu. Deep-learning-enhanced human activity recognition for internet of healthcare things. *IEEE Internet of Things Journal*, 7(7):6429–6438, 2020.

[125] World Health Organization (WHO). Global brief for world health day 2012, good health adds life to years. `http://whqlibdoc.who.int/hq/2012/WHO_DCO_WHD_2012.2_eng.pdf`. Accessed: 2022-08-06.

[126] J. Xiang. *Transfer Learning of Image Classification with Deep Learning Architectures*. PhD thesis, Acadia University, Wolfville, Nova Scotia, Canada, 2015.

[127] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding and clustering analysis. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 478–487, New York City, New York, USA, 19-24 June, 2016.

[128] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *ICML'16: Proceedings of the 3rd international conference on international conference on machine learning*, volume 48 of *ICML*, pages 478–487, 2016.

[129] B. Xu, S. Furao, and J. Zhao. A density-based competitive data stream clustering network with self-adaptive distance metric. *Neural Networks*, 110:141–158, 2019.

[130] X. Yu, H. Li, Z. Zhang, and C. Gan. The optimally designed variational autoencoder networks for clustering and recovery of incomplete multimedia data. *Sensors*, 19:809, 2019.

[131] Zhiwen Yu, Peinan Luo, Jane You, Hau-San Wong, Hareton Leung, Si Wu, Jun Zhang, and Guoqiang Han. Incremental semi-supervised clustering ensemble for high dimensional data clustering. *IEEE Trans. on Knowledge and Data Engineering*, 28(3):701–714, 2015.

[132] H. Zhang, X. Xiao, and O. Hasegawa. A load-balancing self-organizing incremental neural networks. *IEEE T. Neur. Net. Lear.*, 25:1096–1105, 2014.

[133] M. Zhang and A.A. Sawchuk. Usc-had: A daily activity dataset for ubiquitous activity recognition using wearable sensors. In *Proceedings of the ACM Conference on Ubiquitous Computing*, pages 1036–1043, Pittsburgh, PA, USA, 5-8 September 2012.

[134] Shibo Zhang, Yaxuan Li, Shen Zhang, Farzad Shahabi, Stephen Xia, Yu Deng, and Nabil Alshurafa. Deep learning in human activity recognition with wearable sensors: A review on advances. *MDPI Sensors*, 22(4):1476, 2022.

[135] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: A new data clustering algorithm and its applications. *Data Min. Knowl. Disc.*, 1:141–182, 1997.

[136] T. Zhao, Z. Wang, A. Masoomi, and J.G. Dy. Streaming adaptive nonparametric variational autoencoder, 2019.

[137] Xiaokang Zhou, Wei Liang, Kevin I-Kai Wang, Hao Wang, Laurence T. Yang, and Qun Jin. Deep-learning-enhanced human activity recognition for internet of healthcare things. *IEEE Internet of Things Journal*, 7(7):6429–6438, 2020.

[138] Xiaojin Zhu. Semi-supervised learning literature survey. *CS Technical Reports*, 2005.

[139] A. Zugaroglu and V. Atalay. Data stream clustering: a review. *Artificial Intelligence Review*, 54:1201–1236, 2020.