Generalizable User-Specific Mobility Representation Learning Using Autoencoders

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

geoinformation engineering ikg eth zurich

MIE lab
Mobility Information Engineering Lab at ETH Zurich

AXON VIBE

Generalizable User-Specific Mobility Representation Learning Using Autoencoders

Master Project

Autumn Term 2023

Geomatics Master

Author:	Juliette Wicki wickij@ethz.ch
Supervisor:	Prof. Dr. Martin Raubal Dr. Yanan Xin Dr. David Jonietz
Submission Date:	12.01.2024

Acknowledgement

I would like to extend my gratitude to my supervisors, Dr. Yanan Xin and Dr. David Jonietz, for their valuable guidance, support, and dedication throughout this project, including their efforts in organizing and conducting a workshop.
I am appreciative of the enduring support from my family and friends. Especially, I want to thank my colleagues Oliver Heisel, Michelle Halbheer, Arno Rüegg, and Fabian Meier for their invaluable machine learning insights and encouragement during this project.

Abstract

Addressing the demand for sustainable travel solutions, Axon Vibe leverages a smartphone-based platform to predict commuters' travel patterns and encourage eco-friendly alternatives. Understanding the pivotal factors shaping individual travel trajectories stands as a key pursuit for the company.
This project focuses on extracting these critical factors and reconstructing travel patterns by employing an Autoencoder (AE) architecture, integrating long short-term memory (LSTM) layers. To enhance mode detection within the latent space, the architecture incorporates a classification framework. The evaluation using t-distributed stochastic neighbor embedding (t-SNE) confirms that the integration of classification improves the intuitive clustering of transport modes within the latent space.

Contents

List of Figures

List of Tables

Acronyms

AE	Autoencoder.
LSTM	long short-term memory.
MSE	mean squared error.
NaN	Not a Number.
RNN	recurrent neural network.
SDG	Sustainable Development Goal.
t-SNE	t-distributed stochastic neighbor embedding.
UN	United Nations.
WGS84	World Geodetic System 1984.

1. Introduction

Climate change has become an undeniable force reshaping our world, emphasizing the urgency of global action. As nations grapple with its repercussions, the United Nations (UN) set forth ambitious Sustainable Development Goals (SDGs) to combat this crisis (United Nations, 2023b). Among these objectives, one pivotal goal revolves around sustainable settlements, including transportation (United Nations, 2023a). This goal emphasizes the need for sustainable solutions within cities, encompassing transportation systems that are environmentally friendly, efficient, and accessible, recognizing the profound impact of transport on urban landscapes and the environment. Transport stands as a significant contributor to greenhouse gas emissions, with a staggering fifth of global CO_2 emissions attributed to this sector alone (see Figure 1.1) (United Nations, 2021). Alarming as it is, an even more concerning statistic emerges: three-quarters of these emissions stem from road transport (Ritchie, 2020).

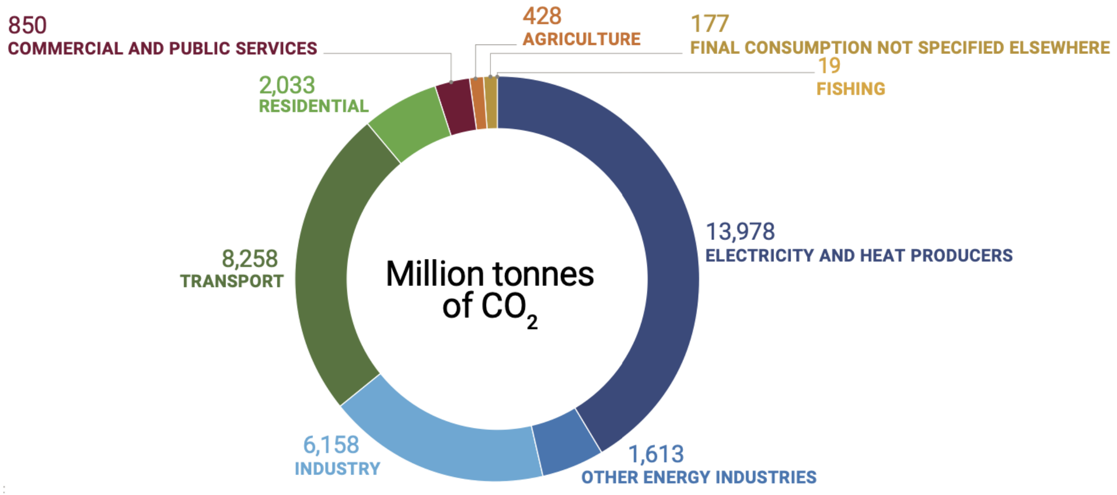


Figure 1.1.: CO_2 emissions by sector (2018), (United Nations, 2021).

In light of these critical challenges, companies worldwide are acknowledging their role in fostering sustainable change. Setting their sights on mitigating this environmental impact, many organizations are taking proactive steps. One such forward-thinking company, Axon Vibe, has made a conscientious commitment: to

revolutionize traffic and make it more sustainable. This dedication aligns with the broader objectives outlined by the UN's sustainability goals, demonstrating a collective effort towards a greener, more sustainable future.
Axon Vibe specializes in providing innovative mobility solutions. Their focus is on leveraging technology to optimize and enhance the efficiency of public transport systems. Axon Vibe's core expertise is the development of intelligent software that utilizes data-driven insights to improve the overall commuter experience. Their smartphone-based platform aims to offer personalized, real-time travel information, helping users navigate public transport networks seamlessly while promoting sustainable and efficient mobility options within cities. Through their innovative approach, Axon Vibe seeks to transform how people interact with and utilize public transport systems, ultimately contributing to more sustainable and accessible urban mobility. Their specific strategy includes not only traffic agencies and commuters but also third-party providers. (Axon Vibe, 2023)
Hence, it is of great interest to the company to learn the key factors driving individual mobility. This project aims to comprehend crucial features from travel information and reduce storage size utilizing the latent space representation of an Autoencoder (AE). In addition, the integration of a classification layer to differentiate between various modes of transport within the latent space is investigated. To unveil hidden insights within the data, the latent space representation with and without classification are examined using t-distributed stochastic neighbor embedding (t-SNE).
The following section (section 2) gives an insight in relevant fields of research and techniques, whereas section 3 focuses on the implementation of the used methods. The data provided by Axon Vibe is presented in section 4. The obtained results are listed and discussed in section 5 and 6 respectively. The last section (section 7) focuses on concluding the paper and presenting an outlook for future investigations.

2. Theory and Related Work

Previous studies have demonstrated that individual mobility traces are highly unique and can be identified using only limited spatiotemporal points (Rákos, Aradi, et al., 2020). Despite the increasing volume of mobility data, studies emphasize the potential to condense salient features into compact representations. Such a compressed, lower-dimensional representation not only reduces memory storage needs but also safeguards individual privacy. Furthermore, compressed lower-dimensional representations can be used for various downstream tasks such as maneuver classification (Rákos, Bécsi, et al., 2021), anomaly detection (Wu and Liang, 2022) or synthetic trajectory data generation (Chen et al., 2021).
This paper aims to use representation learning with an Autoencoder to comprehend key factors driving individual mobility. The compressed representation is examined to efficiently detect the mode of transport. Furthermore, the integration of classification is analyzed to verify whether better mode detection is possible.

2.1. Autoencoder

Autoencoders (AEs) stand out as an effective technique for representation learning. Their primary goal is twofold: reconstructing input data while concurrently learning a condensed representation. The unsupervised neural network model comprises two fundamental components: the encoder and the decoder (Shah and Ganatra, 2022). The encoder is responsible for compressing the input into a lower-dimensional representation by extracting its essential features. Meanwhile, the decoder reconstructs the input using this condensed, encoded representation. This lower-dimensional representation is commonly known as the latent space (Chen et al., 2021).
The loss function serves as a measure of how well the AE model is able to reconstruct its input data. It quantifies the difference between the input data and the output generated by the AE (Sharma, 2023). By minimizing this loss function during the training process, the AE adjusts its parameters to improve its ability to reconstruct the input faithfully. Generally, the more effectively the AE is able to reconstruct the input, the more meaningful the features in the latent space tend to be. Therefore, the latent space representation can be used more effectively to represent the input.

The choice of the loss function is depending on the input type. Commonly used loss functions for AE include mean squared error (MSE) to measure the average squared difference between the input and the output or binary cross-entropy for binary input and output values (Bandyopadhyay, 2021).

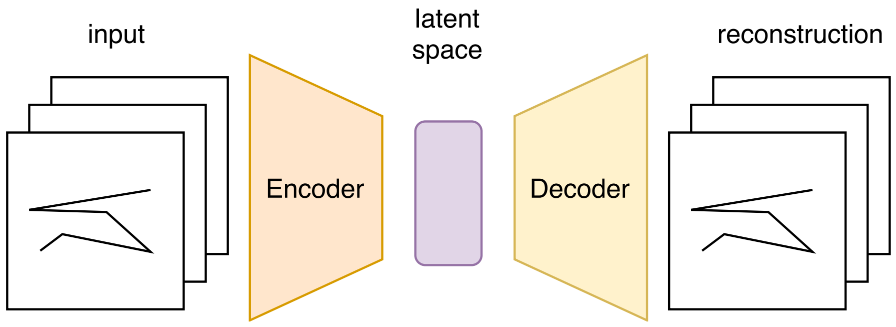


Figure 2.1.: General architecture of an AE.

Figure 2.1 shows a typical architecture of an AE, featuring the encoder, the decoder and the latent space. There exist many extensions to this general network structure for various input types and purposes, such as convolutional AE for image data, denoising AE to remove noise or variational AE including a probabilistic component (Sharma, 2023). For sequential data, AEs with LSTM-layers are widely used (e.g. Graves (2013) and Chen et al. (2021)). The long short-term memory (LSTM) concept, pioneered by Hochreiter and Schmidhuber (1996), can address temporal characteristics, capturing long-term dependencies within time series data by effectively combating vanishing or exploding gradients present in regular recurrent neural network (RNN) models.
The present project proposes an AE with LSTM-layers to learn the latent space representation of individual travel data, described in section 4. Implementation details are listed in section 3.1.

2.2. Classification

Classifying data in different categories is a fundamental task in data analysis and machine learning. Classification algorithms generalize from known data patterns to accurately classify new, unseen data. These classifications are categorical by nature, representing discrete and usually unordered categories such as various types of fruit, colors, or modes of transport. Classification enables automated decision-making, enhances information retrieval, and aids in understanding complex data structures by organizing them into meaningful categories. (Han et al., 2012)

In the present project, classification is used to enrich the AE architecture (see section 3.2). It is analyzed whether the integration of classification leads to more distinct clustering in the latent space with respect to the different modes of transport.

2.3. t-SNE

Since the introduction of t-distributed stochastic neighbor embedding (t-SNE) by Maaten and Hinton (2008), the technique gained popularity in machine learning and data visualization. It is based on dimensionality reduction, aiming to preserve the significant, high-dimensional structure as good as possible (Maaten and Hinton, 2008). It is particularly effective for visualizing high-dimensional data in a lower-dimensional space (usually 2D or 3D) while preserving the local structure and relationships, such as clusters, between data points.
To visualize whether integrating classification in the model architecture leads to enhanced clustering in the latent space, t-SNE is applied in this project to the derived latent spaces.

3. Methodology

The following sections describe the implementation of the techniques introduced previously (see section 2). The code is written in Python using the packages `NumPy` (Harris et al., 2020) for general operations, `keras` (Chollet et al., 2015) and `TensorFlow` (Abadi et al., 2015) for the model architecture, `scikit-learn` (Pedregosa et al., 2011) for t-SNE as well as `matplotlib` (Hunter, 2007) for visualizations.

3.1. Autoencoder

As mentioned in section 2.1, an AE with LSTM-layers is implemented. The LSTM can effectively learn temporal characteristics and is therefore suitable for the mobility data at hand. Individual travel data is split into fixed length sequences (see section 4 for insights in the data structure and preprocessing) to train and test the model. The model architecture is mainly inspired by the following sources: Oliveira (2020) and TensorFlow (2023).
As Figure 2.1 shows, the AE is built of two parts, the encoder and the decoder (see Figure 3.1). They are implemented as distinct sequential models, facilitating the potential extension to a variational AE. The total trainable parameters amount to 62'722, with the encoder comprising 32'376 parameters and the decoder 30'346 parameters (see Table 3.1).

Model (type)	*Output shape*	*Number of parameters*
encoder (sequential)	(None, 5, 5)	32'376
decoder (sequential)	(None, 5, 10)	30'346

Table 3.1.: General model summary of the implemented AE architecture.

The encoder incorporates three LSTM-layers (see Table 3.2), each configured to return sequences due to the requisite format for the LSTM-input of the subsequent layer. Meanwhile, the decoder employs two LSTM-layers alongside a time distributed layer (see Table 3.3). The time distributed layer enables the consistent application of a dense layer, with identical weights, across every time step in the sequence (Brownlee, 2019).

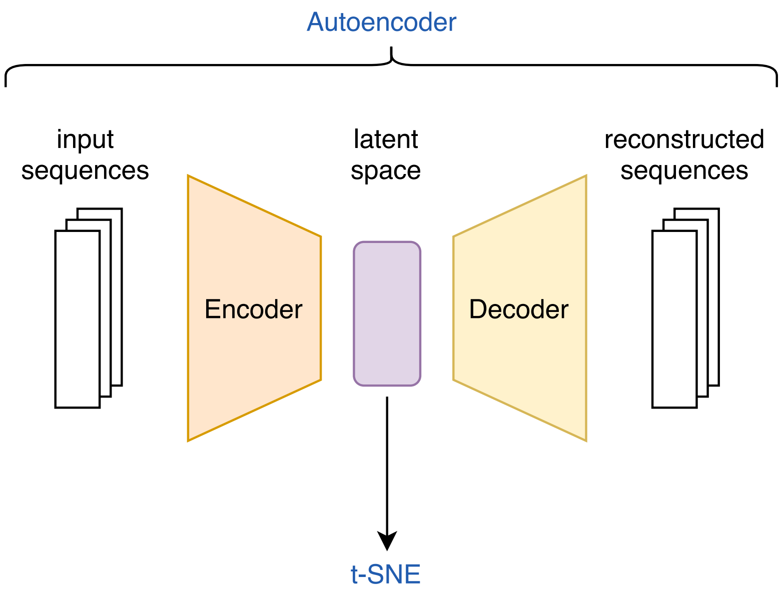


Figure 3.1.: Model Architecture of the AE without classification.

Layer ID	*Layer type*	*Output shape*	*Number of parameters*
1	LSTM-Layer	(None, 5, 64)	19’200
2	LSTM-Layer	(None, 5, 32)	12’416
3	LSTM-Layer	(None, 5, 5)	760

Table 3.2.: Overview of the layer structure used for the encoder.

Regarding the different output shapes listed in Table 3.1, 3.2 and 3.3, the first dimension indicates sequential processing of individual samples, respectively fixed length sequences. The number of time steps per sequence is denoted by the second dimension. The third dimension depends on the choice of hidden units or the latent space and output dimension. The last is represented by the decoder output and matches the input size to fulfill the reconstruction task. The determination of the number of hidden units and the latent space dimension is based on empirical selection.

Employing the Adam optimizer with a learning rate of 0.0001, the model undergoes a maximum of 5’000 epochs during training, incorporating a validation split of 0.1. Training is designed to cease if the loss diminishes or encounters a NaN (Not a Number) value. The MSE serves as the chosen loss function for this model. To expedite the training process and optimize efficiency, a batch size of 128 is set. Table A.1 in the appendix A.1 provides an overview of all hyperparameters and

Layer ID	*Layer type*	*Output shape*	*Number of parameters*
1	LSTM-Layer	(None, 5, 32)	4’864
2	LSTM-Layer	(None, 5, 64)	24’832
3	Time Distributed Dense Layer	(None, 5, 10)	650

Table 3.3.: Overview of the layer structure used for the decoder.

configurations used.
The outcomes produced by this model, specifically the plot of the loss function across the epochs and the plot comparing an input sequence to its reconstruction, are listed in section 5.1. They allow assessing the performance of the model.

3.2. Classification

A second model architecture is designed, building upon the same encoder and decoder configuration employed in the initial AE. This augmented architecture integrates a classifier following the AE (see Table 3.4 and Figure 3.2). Hence, the total parameters for the encoder and decoder remain consistent with the previous AE model. The classification segment introduces 7’818 parameters additionally, resulting in a combined total of 70’540 trainable parameters.

Model (type)	*Output shape*	*Number of parameters*
encoder (sequential)	(None, 5, 5)	32’376
decoder (sequential)	(None, 5, 10)	30’346
classifier (sequential)	(Number of sequences, 10)	7’818

Table 3.4.: Model summary of the implemented AE architecture integrating the classification. The dimension of the classifier is depending on the data in use (see section 4.2).

The classifier comprises two dense layers, separated by a dropout layer set at a rate of 0.5 (see Table 3.5). The first dense layer contains 128 hidden units, while the subsequent dense layer consists of 10 units to align with the input data. These parameters are set based on empirical evaluation. The labels for this classification task are derived from the modes of transport found in the vehicle truth file (see section 4).
The hyperparameters for this model architecture mirror those used in the AE without the classifier, maintaining uniformity in configurations. One disparity

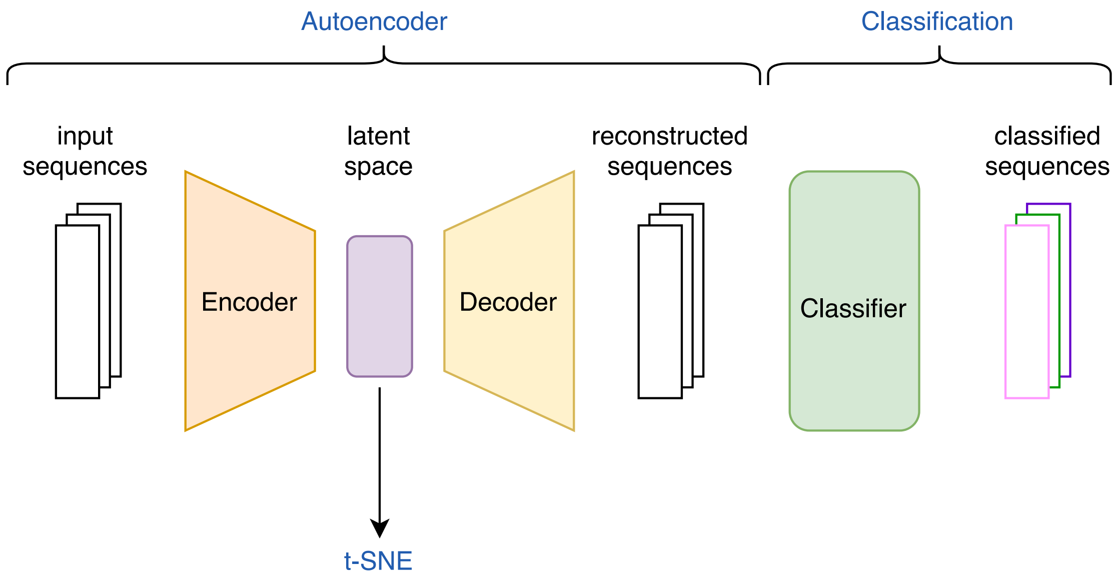


Figure 3.2.: Model Architecture of the AE with classification.

Layer ID	*Layer type*	*Output shape*	*Number of parameters*
1	Dense	(Number of sequences, 128)	6'528
2	Dropout	(Number of sequences, 128)	0
3	Dense	(Number of sequences, 10)	1'290

Table 3.5.: Overview of the layer structure used for the classifier.

lies in the choice of the loss function; here, the categorical cross-entropy loss is employed specifically for the classification task (see Table A.1 in the appendix A.1). Furthermore, class weighting is applied to cope with the imbalanced data as the number of sequence of certain modes highly outnumbers the sequences of other modes (Igareta, 2021).

Besides plotting the loss function across the epochs, a confusion matrix is produced to evaluate the performance of this model (Kulkarni et al., 2020). It allows to understand the performance of the model by presenting a summary of the predicted classes versus the occurrence of the respective classes. The generated figures are listed in section 5.2.

3.3. t-SNE

To illustrate the impact of integrating classification in the model architecture, t-SNE is applied to the latent space derived by both models. Hence, the model architecture depicted in Figure 3.1 is applicable to the AE without classification, while Figure 3.2 represents the model architecture for the AE with classification.

4. Data

The company Axon Vibe collects user-specific mobility data using a smartphone-based platform (Axon Vibe, 2023). The corresponding data structure and necessary preprocessing for this project are described in this section. As the data is confidential, it is not shared together with this report.

4.1. Data Structure

For the present project, only quality assured data, so-called field tests, provided by Axon Vibe, are used. However, the structure described hereafter is valid for the data without quality assurance as well.

For each field test, four different data sets exist. One of the data sets contains general information about the tracked journeys, which is used to filter and extract the data in the other files. The other data sets store recorded sensor data and are used to train and test the model architecture. Table 4.1 and the following subsections give a brief overview of the files and their data used within this project.

Data set	*Features*	*Usage*
Vehicle Truth	Identification code, start and end time of trip, true mode label	Extraction of sensor data, true labels for classification
Pedometer	Number of steps	Training and testing the architecture
Motion	Detected mode label	Training and testing the architecture
Location	Latitude, longitude, horizontal accuracy	Training and testing the architecture

Table 4.1.: Summary of the data sets, their features of interest and usage within the project.

4.1.1. Vehicle Truth

This file has one record per tracked journey, revealing the general information of one trip, called *leg* by Axon Vibe. Namely information about the user, the start and end time, start and end location, the operating system of the smartphone in use, as well as the true vehicle type are stored. The last one is a categorical variable with ten possible values: ferry, subway, rail, car, tram, cycle, walking, funicular, bus or other. The user information, or more precisely the information of the device, as one user can have multiple devices, is encoded in an identification code. Together with the start and end time of the journey, this code is used during preprocessing to filter the sensor data (see section 4.2). The true vehicle type serves as the label utilized for classification (see section 3.2). In general, the operating system is not used for this project but reveals interesting insight into the data collection of the various devices.

4.1.2. Pedometer

The data recorded with the pedometer sensor is stored in this file. The pedometer counts the number of steps within a certain interval. Besides the number of steps and the interval, the identification code and the operating system of the device are tracked. However, the sampling rate, and therefore the interval, is irregular and strongly depending on the operating system. Apple devices record the counted steps every minute if the sensor is active. Android devices, on the other hand, sample with a higher frequency, approximately every 5 seconds, if the sensor is active. If no steps are tracked, both type of devices have a lower sampling rate, sampling approximately every 3 minutes.

4.1.3. Motion

The motion file contains information provided by the device about the detected mode as Boolean values. The devices can state whether one is stationary, walking, running, cycling, automotive or whether the method of locomotion is unknown. Furthermore, it stores the confidence level in percentage, the timestamp, the identification code and the operating system of the device.

4.1.4. Location

The location tracked by the device is stored in another separate file. It provides information about various aspects such as latitude and longitude, horizontal and vertical accuracy, speed, bearing and altitude. For this project, only the latitude and longitude in WGS84 and the horizontal accuracy in meters are of interest. The identification code, the operating system of the device and the timestamp are recorded as well.

4.2. Preprocessing

Preprocessing is a pivotal step aimed at transforming the raw data into the format which aligns with the requirements of the model architecture. Besides extracting and merging information from the four different data sets, the data must be split into sequences of fixed length and normalized for stability. Hereafter, the preprocessing is described for one field test composed of four files as described in section 4.1. If the preprocessing, and hence the training and testing of the architecture, should be done with several field tests, the procedure must be repeated for each individual field test. It is not recommended to combine field tests across different geographical regions due to the distinct regional characteristics that may introduce complexities impeding the model's learning process.
Figure 4.2 summarizes the preprocessing, implemented in a Python script using the packages `NumPy` (Harris et al., 2020), `pandas` (McKinney, 2010) and `scikit-learn` (Pedregosa et al., 2011), graphically.

4.2.1. Loading Data

The data preprocessing starts with loading the respective four files of a single field test. While loading, the fields containing timestamps are converted to a date format to simplify the filtering later on. The detected modes in the motion file are switched from Boolean values to 0 and 1 so that one-hot encoding applies. In general, one-hot encoding is used to represent categorical values numerically. It generates for each possible category a new binary feature which is set to 1 if the present data point belongs to this category (see Table 4.2 for an example) (Manai et al., 2023). Additionally, one-hot encoding is implemented on the true vehicle types from the vehicle truth file to create one-hot encoded labels. It must be considered that maybe not all possible vehicle types are represented in the selected field test which is why the one-hot encoding is based on a master list containing all possible types.

stationary	*walking*	*running*	*cycling*	*automotive*	*unknown*
0	1	0	0	0	0

Table 4.2.: Example of one-hot encoding for the detected modes in the motion file; here the detected mode is walking.

4.2.2. Sequencing and Binning

As mentioned in section 4.1, the vehicle truth file has one entry per recorded leg. The legs vary in duration, which is why they are split into fixed length sequences of five minutes. The constructed sequences overlap as the start of the next sequence is shifted by one minute to generate more data. Within the sequences, bins of one minute are created to simulate a regular sampling rate. If one leg is shorter than five minutes, it is prolonged using default values to still fulfill the requirement of fixed length sequences. Figure 4.1 visualizes the sequencing and binning of the legs. The duration of the sequences and bins are chosen empirically and may need to be adapted for other use cases.

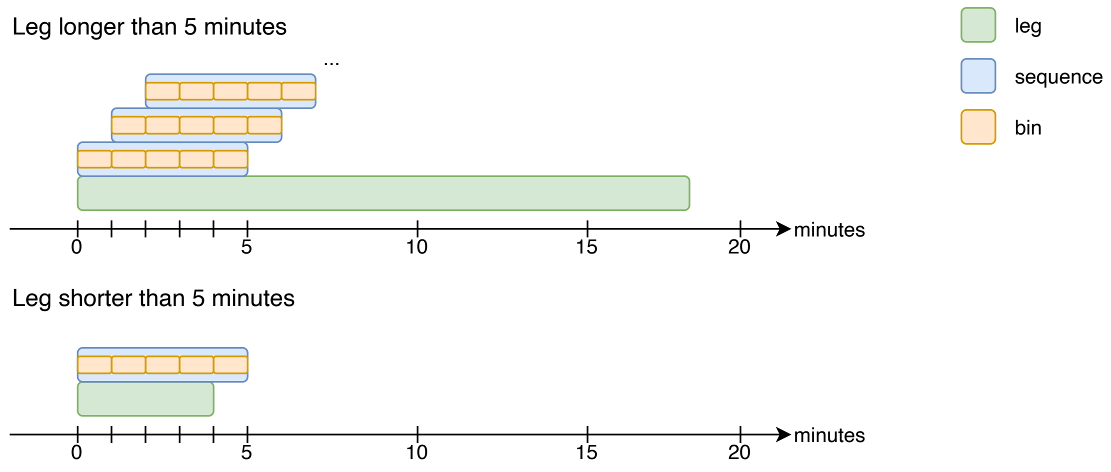


Figure 4.1.: Illustration of sequencing and binning.

4.2.3. Data Extraction

For each sequence, respectively bin, the data is extracted from the other three files. Since the sensors continuously record data, irrespective of an ongoing leg, filtering this sensor data becomes necessary. To filter the data, the identification code and the start and end time of the sequence are used. Depending on the sensor data, different additional steps are required, outlined below.

For the pedometer data, the number of steps are extracted and summed within one bin. If no data is available for one bin, the default value 0 is assigned. The beginning and end of a leg are treated specially to include important walking activities closely preceding and following a trip. As such, a five-minute window before and after the leg is averaged and incorporated into the initial and final bins of the first and last sequences, respectively.
Regarding the motion data, the one-hot encoded mode with the highest confidence level is chosen for each bin. In cases where a bin lacks data, a default vector comprising zeros is employed.
To aggregate the location information per bin, the median is taken. If data is unavailable, a default vector consisting of zeros is utilized.
The extracted data from the three sensor files, also called features, are merged if they not only contain default values. This results in a 3-dimensional matrix with dimensions delineating the number of sequences, bins, and features. The count of sequences varies depending on the number and duration of legs in a field test. Conversely, the count of bins remains constant at 5, attributed to the segmentation of 5-minute sequences into 1-minute bins. Correspondingly, the number of features stands at a constant value of 10, arising from distinct sensor data aspects. Among these, one feature corresponds to the pedometer output denoting steps, while six features stem from motion data, encapsulated via one-hot encoding representing detected modes. The remaining three features pertain to location data encompassing latitude, longitude, and horizontal accuracy.

4.2.4. Label Generation

The previously one-hot encoded vehicle type of the leg is repeated to get a label for each sequence, resulting in a 2-dimensional matrix. The dimensions are based on the number of sequences and labels. The first dimension should align the first dimension of the extracted data. The number of labels is constantly 10, mirroring the ten possible labels.

4.2.5. Normalization and Splitting

The outlined process needs to be repeated for all field tests of interest.
In a next step, all features which are not one-hot encoded, namely the ones derived by the pedometer and location files, are normalized using a min-max scaler. To evaluate and validate the model's performance, the preprocessed data and labels are split into a training and test set. These sets are then stored separately, enabling their reuse without the necessity of rerunning the entire process.

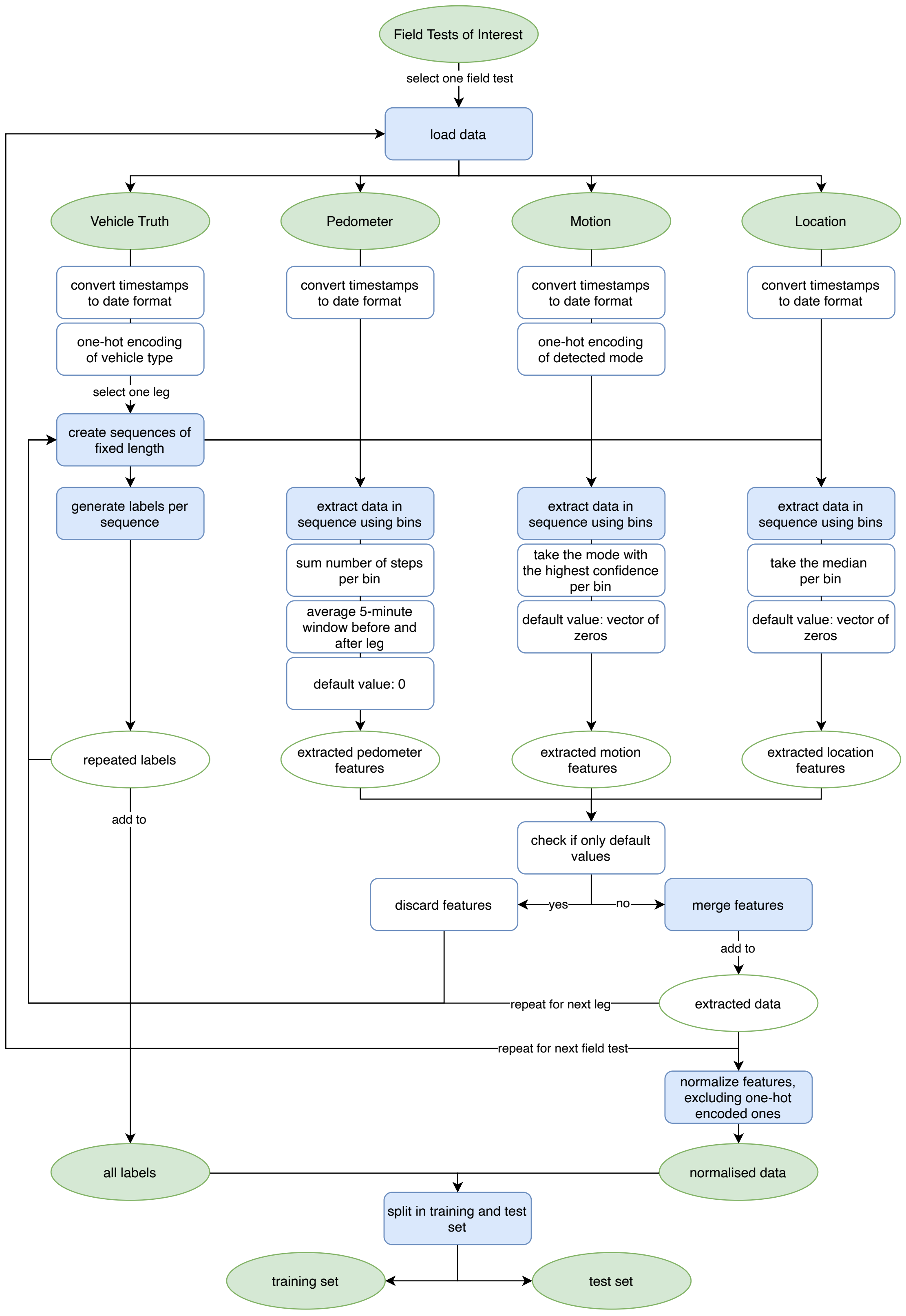

Field Tests of Interest
select one field test
load data
Vehicle Truth
Pedometer
Motion
Location
convert timestamps to date format
convert timestamps to date format
convert timestamps to date format
convert timestamps to date format
one-hot encoding of vehicle type
one-hot encoding of detected mode
select one leg
create sequences of fixed length
generate labels per sequence
extract data in sequence using bins
extract data in sequence using bins
extract data in sequence using bins
sum number of steps per bin
take the mode with the highest confidence per bin
take the median per bin
average 5-minute window before and after leg
default value: vector of zeros
default value: vector of zeros
default value: 0
repeated labels
extracted pedometer features
extracted motion features
extracted location features
add to
check if only default values
discard features
yes
no
merge features
add to
repeat for next leg
extracted data
repeat for next field test
normalize features, excluding one-hot encoded ones
all labels
normalised data
split in training and test set
training set
test set

Figure 4.2.: Illustration of data preprocessing; the main steps and outputs are highlighted.

5. Results

Both models described in section 3 are trained utilizing the dedicated training set, and their performance is subsequently evaluated using the separate test set comprising unseen data. The forthcoming sections unveil the outcomes achieved by these models, shedding light on their respective performances and insights derived from the evaluation process. The obtained results are subject to slight variations across different runs of the model due to the stochastic nature of the training procedure.

5.1. Autoencoder

The model evaluation revolves around two key aspects: the loss function analysis and visual comparison between original samples and their reconstructed counterparts.
The loss function is decaying both for the training and the validation set, as can be seen in Figure 5.1.

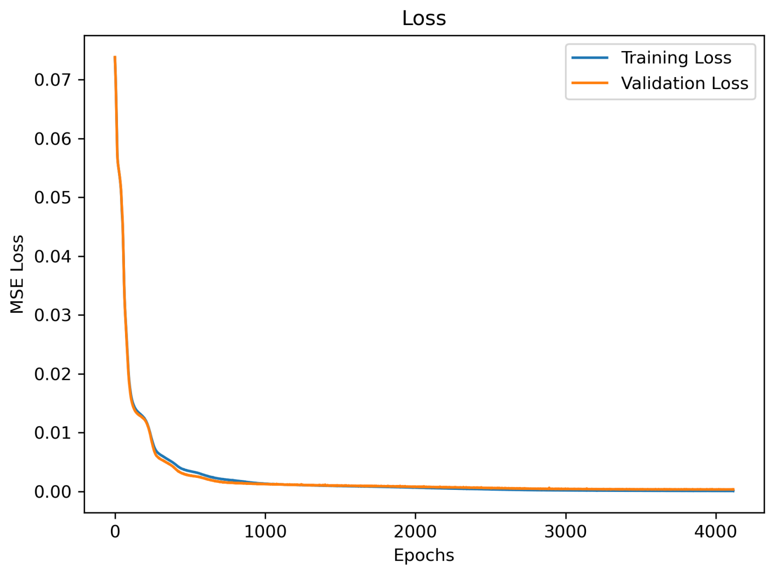


Figure 5.1.: Training and validation loss of the AE model without classification.

To plot the original sequence and the reconstruction, each of the ten features is represented in a separate subplot. Given the normalization of data, all values fall within the range of 0 to 1. Figure 5.2 shows an original sequence from the test set juxtaposed with its reconstruction. This comparison allows for a direct observation of the model's ability to recreate the original sequence, serving as a visual gauge for its performance.
The reconstruction of a sequence from the training set is included in the appendix (see Figure A.2 in the appendix A.2).

5.2. Classification

Similar to the model without classification, the assessment of model performance relies on the loss function as a primary measure. The loss, including the training and validation loss, is generally decreasing (see Figure 5.3).
Additionally, a confusion matrix is employed to compare predicted and actual classes. Within this matrix, numbers accompanied by colors represent the occurrence of various combinations, offering a visual and quantitative insight into the model's predictive accuracy. Row-standardization of this matrix provides a clear depiction of the classification's efficacy, showcasing how well the model discerns and predicts different classes. In Figure 5.4, the confusion matrix illustrates the distribution of predictions against actual classes within the test set. For the confusion matrix pertaining to the training set, kindly refer to Figure A.1 in the appendix A.2.

5.3. t-SNE

As mentioned in section 3.3, t-SNE is used to visualize the latent space derived by the models. Figure 5.5 displays the latent space visualization for the model without classification, while Figure 5.6 shows the latent space representation for the model incorporating classification.

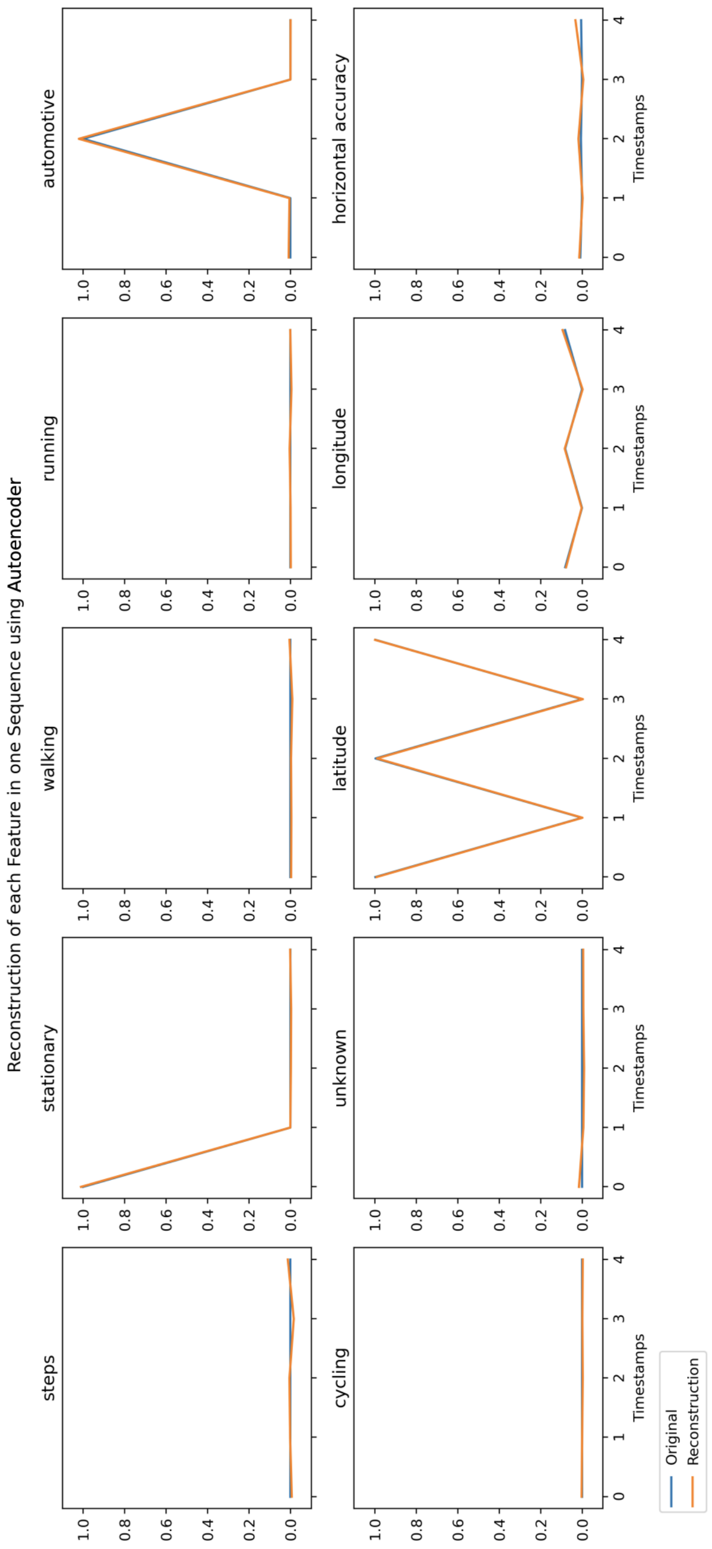


Figure 5.2.: One original sequence of the test set and its reconstruction with the respective features as subplots.

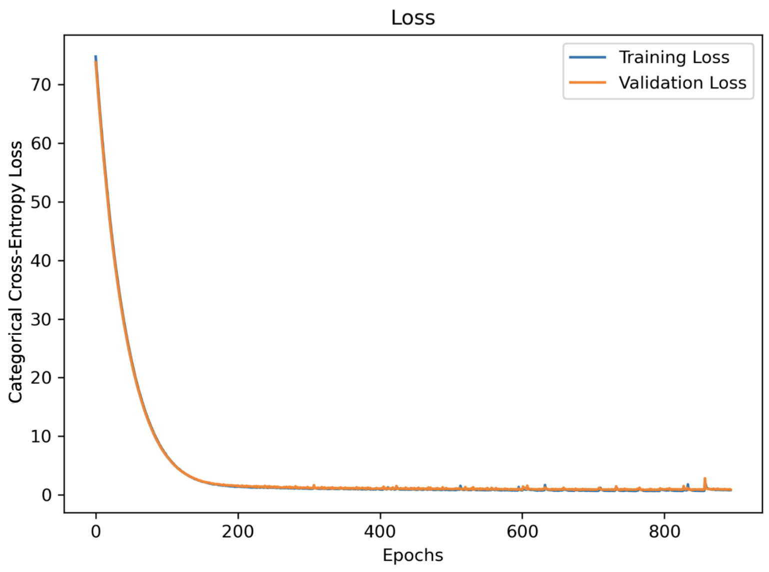


Figure 5.3.: Training and validation loss of the AE model with classification.

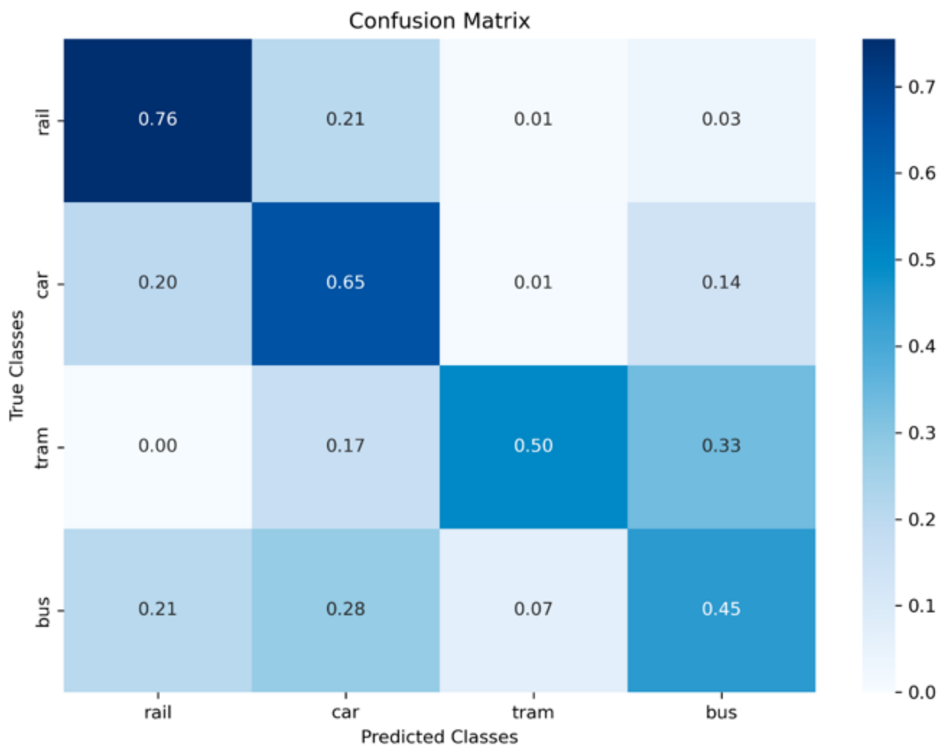


Figure 5.4.: Confusion matrix of the test set.

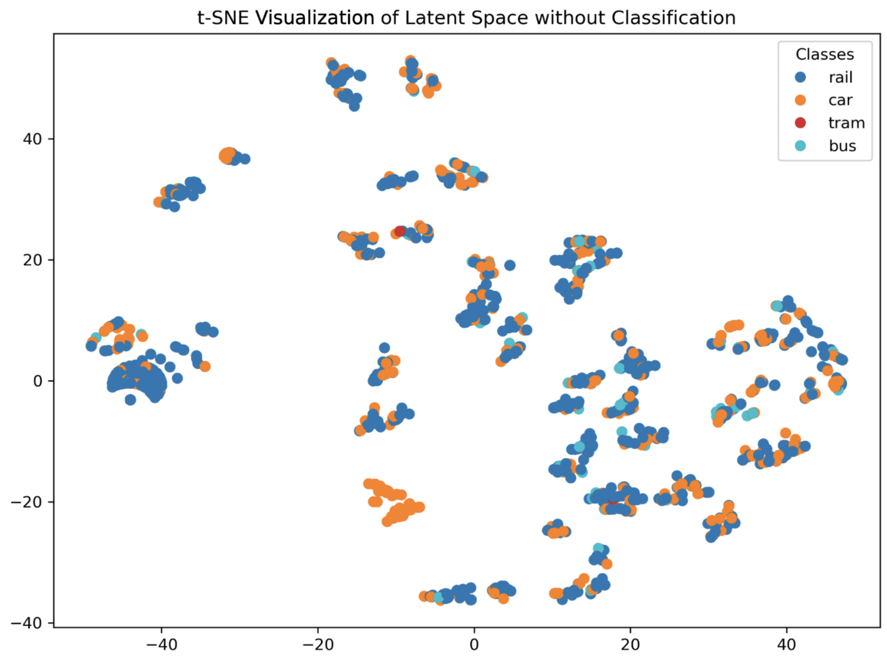


Figure 5.5.: Visualization of the latent space for the AE without classification using t-SNE.

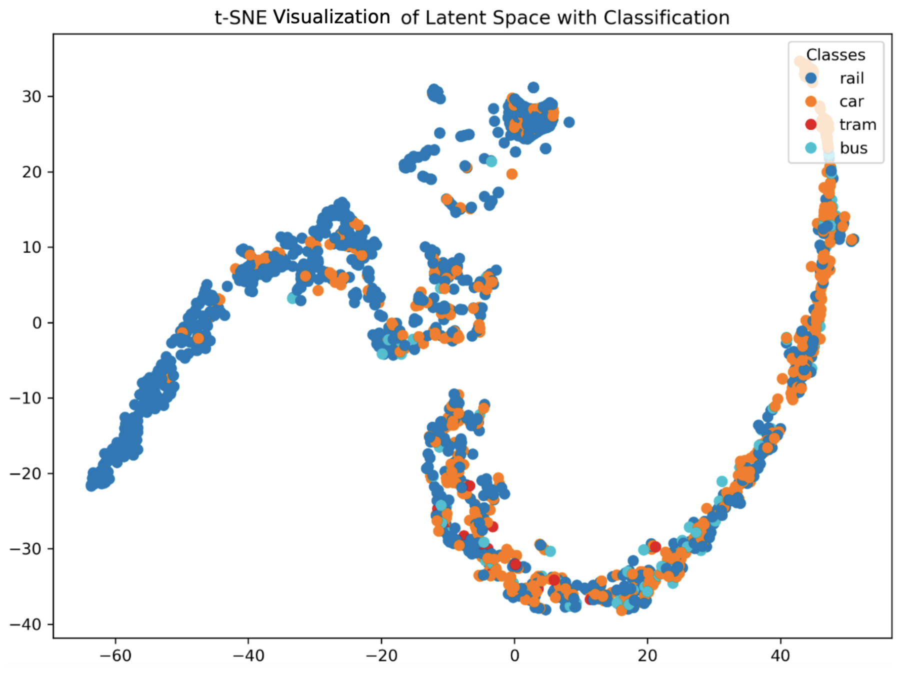


Figure 5.6.: Visualization of the latent space for the AE with classification using t-SNE.

6. Discussion

This chapter focuses on the interpretations, implications, and significance of the findings listed in section 5.

6.1. Autoencoder

As both the training and validation losses decrease and converge to a stable value (see Figure 5.1), it generally suggests that the model is learning the underlying patterns in the data without overfitting. This scenario is ideal as it indicates that the model has learned to generalize from the training data to new, unseen data in the validation set.
Additionally, the early stopping criteria, determined by the loss function, is satisfied as the model converges in fewer than 5'000 epochs. This convergence further supports the model's ability to learn efficiently within a reasonable training period.
The model's capacity to generalize is further evidenced in its reconstruction capabilities. It is able to reconstruct sequences within the test set (see Figure 5.2), showing its adaptability across different features – even accommodating unconventional data like the latitude information of this specific sequence. Notably, the model showcases proficiency in learning the combination of normalized and the one-hot encoded features.
Therefore, the AE model with LSTM-layers adeptly learns a latent space representation that accurately captures the essence of the original mobility data, suitable for subsequent reconstruction purposes.

6.2. Classification

The loss function in the model, when compared to the AE without classification, demonstrates similarity. Both training and validation losses display a consistent reduction (see Figure 5.3), indicating a strong model performance in learning and generalizing from the data. Additionally, the model's behavior exhibits slightly more fluctuations than a sole focus on reconstruction, representing its ability to escape a local minimum.
During training fewer epochs are required, which shows that classification is significantly easier than reconstruction.

Further evaluation through the confusion matrix reaffirms the effectiveness of the classification aspect (see Figure 5.4). Although some misclassifications persist, the model's accuracy exceeds 60 percent for half of the classes. However, the model faces challenges in learning classes with fewer samples, such as tram and bus in this specific field test. Trams are often misclassified as buses, whereas buses tend to be misclassified as rail or car. Meanwhile, mispredictions between car and rail occur at a relatively similar frequency. Despite these nuances, a majority of the predictions tend to align along the diagonal, indicating accurate classifications.

6.3. t-SNE

t-SNE is applied to the latent space representation of both models to visualize whether classification yields clearer clustering in the latent space regarding the modes of transport.
The initial hypothesis posits that integrating classification into the architecture would not enhance clustering within the latent space. However, upon examining Figures 5.5 and 5.6, this hypothesis is disproven. Figure 5.5, showcasing the AE without classification, lacks evident clustering within the latent space. In contrast, Figure 5.6, while not displaying distinct cluster boundaries, exhibits relatively clearer and more intuitive clustering. Points of the same color, denoting the same mode of transport, tend to cluster more prominently. This observation leads to the falsification of the initial hypothesis, indicating that integrating classification indeed yields a more clustered latent space representation compared to using only the AE.

7. Conclusion and Outlook

This project delves into the potential of AEs when applied to mobility data, exploring their efficacy in representation learning. Especially, the inclusion of classification within the AE architecture to cluster the latent space unveils distinct benefits. The study establishes a foundational understanding, demonstrating the feasibility of employing AEs for both reconstruction and effective representation learning. The incorporation of classification, enhancing clustering within the latent space, underscores the potential of this model architecture.
While the focus of this study primarily centers on a specific field test, it sets the stage for broader investigations in this domain. An analysis aimed at determining the suitability of different devices for collecting the data presents a promising avenue for exploration. Moreover, future endeavors could expand the model's application to different regions featuring diverse spatial characteristics. Enhancements to the architecture, particularly to enable real-time data processing, stand as a crucial prospect for future research projects.
As a next step, adapting the existing architecture or combining proposed models offers an intriguing trajectory. One avenue involves leveraging a weighted loss framework, integrating MSE for reconstruction and categorical cross-entropy for classification, thereby fine-tuning the model's performance. An adaptation towards a supervised AE model (see Le et al. (2018)) by implementing classification directly from the latent space while regulating both losses presents an exciting refinement. Furthermore, extending the model to a variational AE by introducing a sampling layer after the encoder could significantly augment its capabilities. The visual inspection of the latent space using t-SNE could be complemented with a clustering approach.
These adaptations of the model architecture, whether implemented in their current state or refined further, hold significant promise for efficient mode detection of new samples. This prospect is particularly appealing for Axon Vibe, showcasing high potential for the company's interests in efficient mobility data representation and mode detection. This work lays a strong foundation for future research and practical applications in the realm of mobility data analysis.

Bibliography

Abadi, M. et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. URL: `https://www.tensorflow.org/`.

Axon Vibe (2023). *Axon Vibe - Proposition*. URL: `https://axonvibe.com/what-we-do`.

Bandyopadhyay, H. (06/2021). *Autoencoders in Deep Learning: Tutorial & Use Cases [2023]*.

Brownlee, J. (08/2019). *How to Use the TimeDistributed Layer in Keras*. URL: `https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/`.

Chen, X. et al. (03/2021). "TrajVAE: A Variational AutoEncoder model for trajectory generation". In: *Neurocomputing* 428, pp. 332–339. DOI: `10.1016/j.neucom.2020.03.120`.

Chollet, F. et al. (2015). *Keras*. https://keras.io.

Graves, A. (08/2013). "Generating Sequences With Recurrent Neural Networks". In.

Han, J., Kamber, M., and Pei, J. (2012). "8 - Classification: Basic Concepts". In: *Data Mining (Third Edition)*. Ed. by J. Han, M. Kamber, and J. Pei. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, pp. 327–391. DOI: `https://doi.org/10.1016/B978-0-12-381479-1.00008-3`. URL: `https://www.sciencedirect.com/science/article/pii/B9780123814791000083`.

Harris, C. R. et al. (09/2020). "Array programming with NumPy". In: *Nature* 585(7825), pp. 357–362. DOI: `10.1038/s41586-020-2649-2`.

Hochreiter, S. and Schmidhuber, J. (1996). "LSTM can Solve Hard Long Time Lag Problems". In: *Advances in Neural Information Processing Systems*. Ed. by M. C. Mozer, M. Jordan, and T. Petsche. Vol. 9. MIT Press. URL: `https://proceedings.neurips.cc/paper_files/paper/1996/file/a4d2f0d23dcc84ce983ff9157f8b7f88-Paper.pdf`.

Hunter, J. D. (2007). "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9(3), pp. 90–95. DOI: `10.1109/MCSE.2007.55`.

Igareta, A. (06/2021). *Dealing with Imbalanced Data in TensorFlow: Class Weights*. URL: `https://towardsdatascience.com/dealing-with-imbalanced-data-in-tensorflow-class-weights-60f876911f99`.

Kulkarni, A., Chong, D., and Batarseh, F. A. (2020). “Foundations of data imbalance and solutions for a data democracy”. In: *Data Democracy*. Elsevier, pp. 83–106. DOI: 10.1016/B978-0-12-818366-3.00005-8.

Le, L., Patterson, A., and White, M. (2018). “Supervised autoencoders: Improving generalization performance with unsupervised regularizers”. In: *32nd Conference on Neural Information Processing Systems*.

Maaten, L. van der and Hinton, G. E. (2008). “Visualizing Data using t-SNE”. In: *Journal of Machine Learning Research* 9(86), pp. 2579–2605.

Manai, E., Mejri, M., and Fattahi, J. (07/2023). “Impact of Feature Encoding on Malware Classification Explainability”. In.

McKinney, W. (2010). “Data Structures for Statistical Computing in Python”. In: pp. 56–61. DOI: 10.25080/Majora-92bf1922-00a.

Oliveira, D. (2020). *Time-series forecasting with LSTM autoencoders*. URL: https://www.kaggle.com/code/dimitreoliveira/time-series-forecasting-with-lstm-autoencoders.

Pedregosa, F. et al. (2011). “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Rákos, O., Aradi, S., Bécsi, T., and Szalay, Z. (09/2020). “Compression of Vehicle Trajectories with a Variational Autoencoder”. In: *Applied Sciences* 10(19), p. 6739. DOI: 10.3390/app10196739.

Rákos, O., Bécsi, T., and Aradi, S. (07/2021). “Adversarial Autoencoder for trajectory generation and maneuver classification”. In: *2021 IEEE 25th International Conference on Intelligent Engineering Systems (INES)*. IEEE, pp. 000013–000018. DOI: 10.1109/INES52918.2021.9512929.

Ritchie, H. (2020). “Cars, planes, trains: where do CO2 emissions from transport come from?” In: *Our World in Data*.

Shah, N. and Ganatra, A. (12/2022). “Comparative Study of Autoencoders-Its Types and Application”. In: *2022 6th International Conference on Electronics, Communication and Aerospace Technology*. IEEE, pp. 175–180. DOI: 10.1109/ICECA55336.2022.10009387.

Sharma, A. (07/2023). *Introduction to Autoencoders*. URL: https://pyimagesearch.com/2023/07/10/introduction-to-autoencoders/.

TensorFlow (2023). *Introduction to Autoencoders*. URL: https://www.tensorflow.org/tutorials/generative/autoencoder.

United Nations (2021). *SUSTAINABLE TRANSPORT, SUSTAINABLE DEVELOPMENT INTERAGENCY REPORT I SECOND GLOBAL SUSTAINABLE TRANSPORT CONFERENCE*. Tech. rep. Department of Economic and Social Affairs.

United Nations (2023a). *Make cities and human settlements inclusive, safe, resilient and sustainable*. URL: https://sdgs.un.org/goals/goal11.

United Nations (2023b). *The 17 Goals*. URL: https://sdgs.un.org/goals#.

Wu, L. and Liang, J. (11/2022). "Anomaly detection based on temporal convolution Autoencoders". In: *Journal of Physics: Conference Series* 2366(1), p. 012041. DOI: 10.1088/1742-6596/2366/1/012041.

A. Appendix

A.1. Hyperparameters

Hyperparameter / configuration	*Definition*
Optimizer	Adam
Learning rate	0.0001
Loss function for AE without classification	MSE
Loss function for AE with classification	Categorical cross-entropy
Maximum number of epochs	5'000
Batch size	128
Hidden units of LSTM-layers in AE	64 and 32, in reversed order for the decoder
Latent space dimension	5
Hidden units of dense layers in classifier	128 and 10

Table A.1.: Overview of hyperparameters and configurations.

A.2. Additional Results

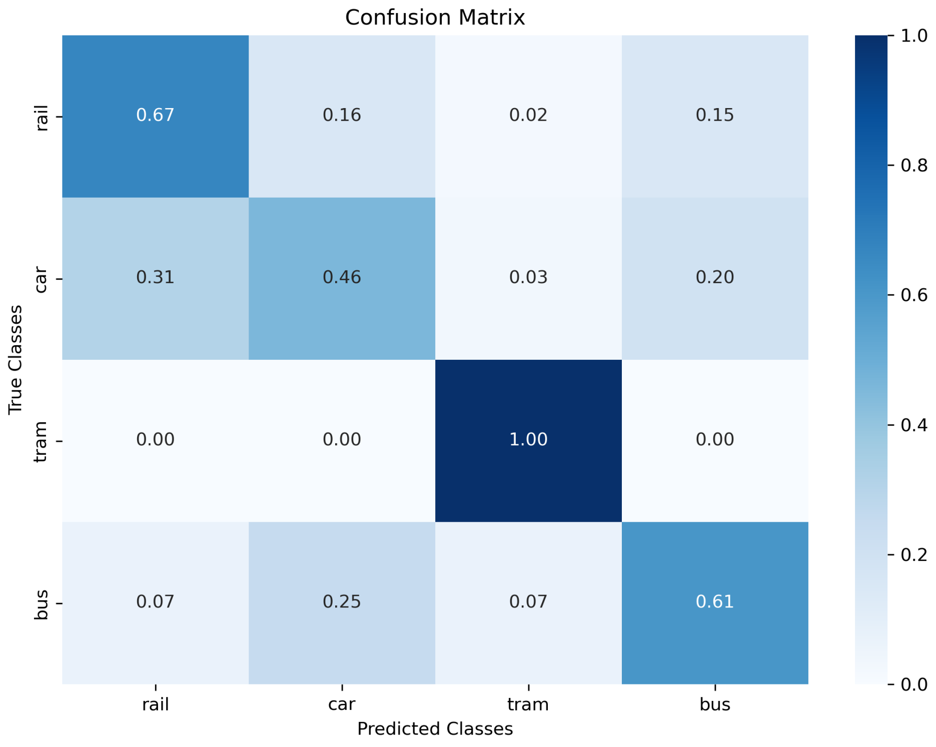


Figure A.1.: Confusion matrix of the training set.

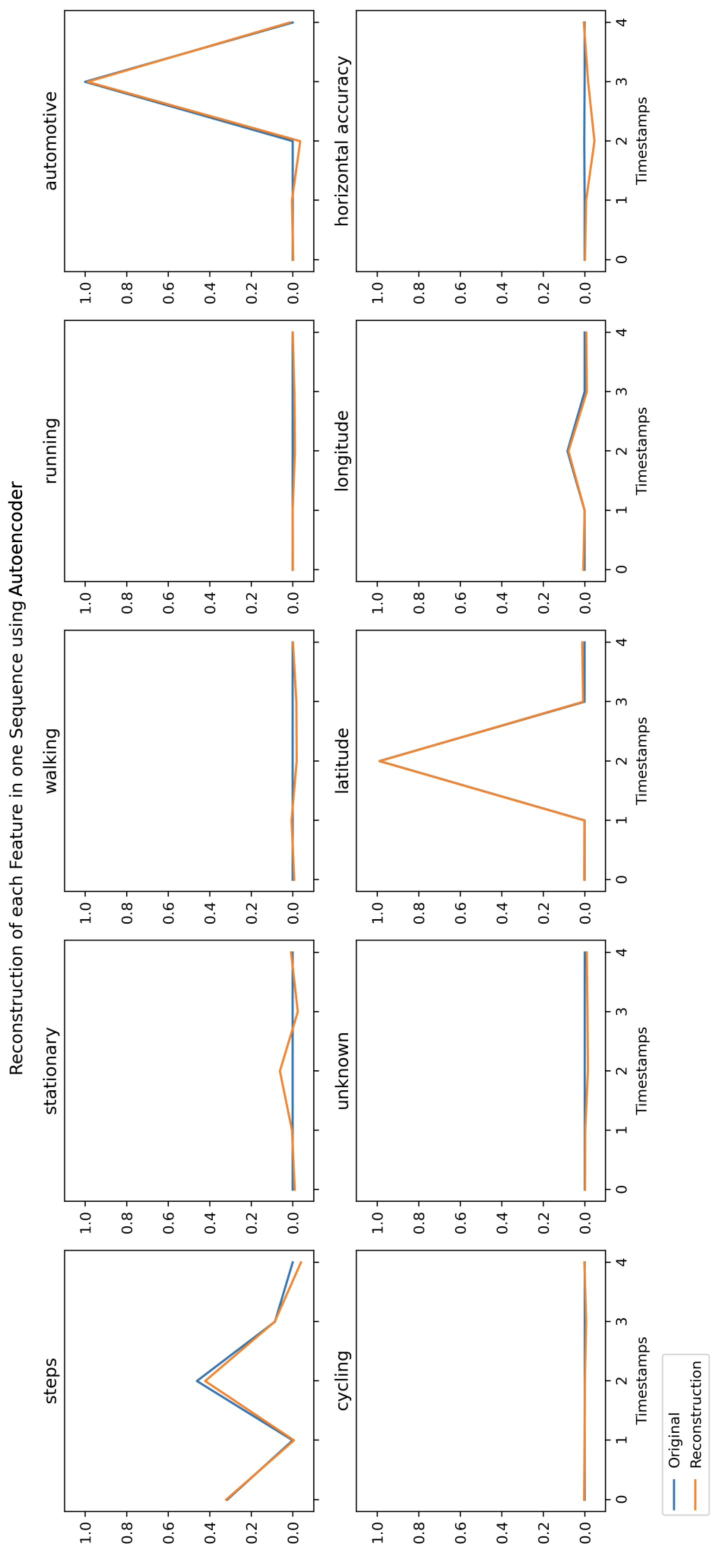
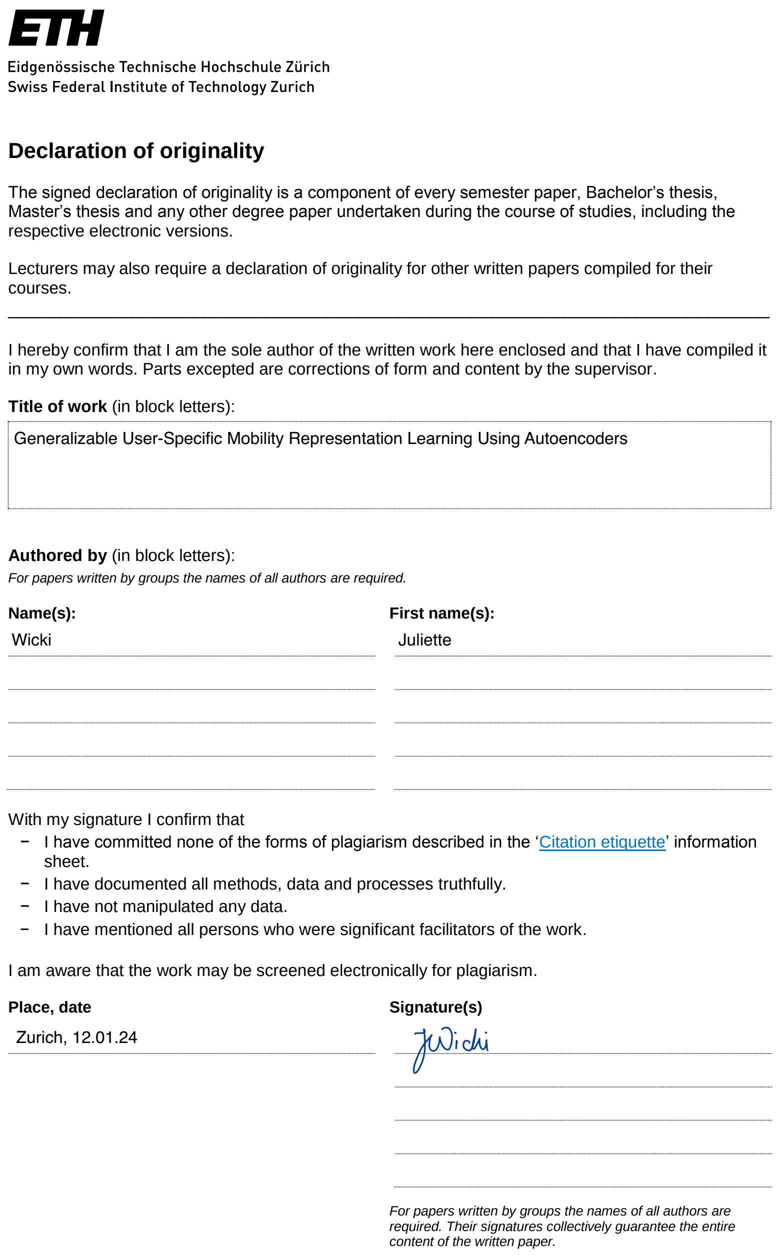


Figure A.2.: One original sequence of the training set and its reconstruction with the respective features as subplots.

Declaration of Originality



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Generalizable User-Specific Mobility Representation Learning Using Autoencoders

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Wicki	Juliette

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 12.01.24	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.