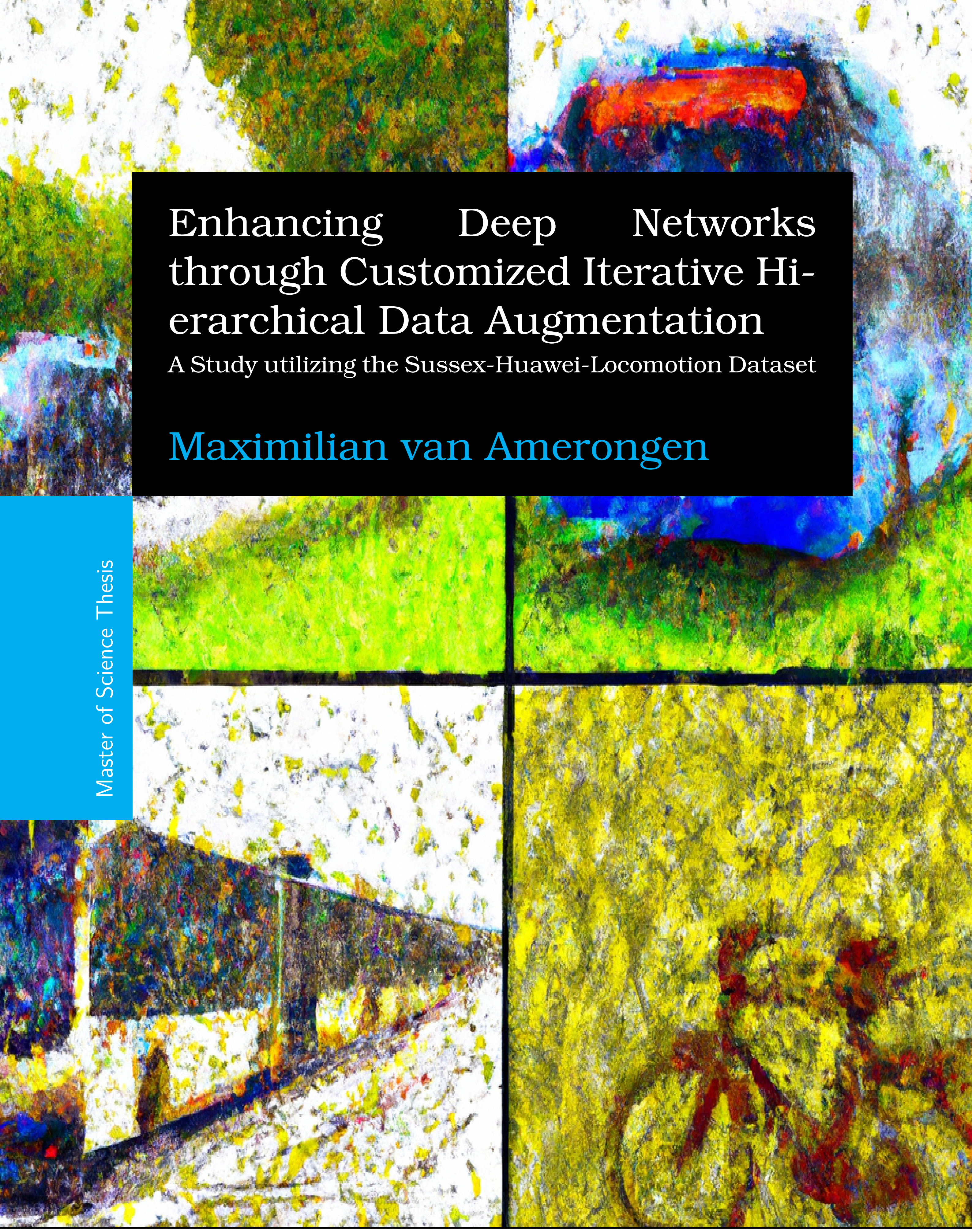

Enhancing Deep Networks through Customized Iterative Hierarchical Data Augmentation

A Study utilizing the Sussex-Huawei-Locomotion Dataset

Maximilian van Amerongen

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

Enhancing Deep Networks through Customized Iterative Hierarchical Data Augmentation

A Study utilizing the Sussex-Huawei-Locomotion Dataset

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Maximilian van Amerongen

August 7, 2023

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

The work in this thesis was supported by the German Aerospace Center (DLR). Their cooperation is hereby gratefully acknowledged.

Abstract

Artificial Neural Networks (ANNs) have emerged as a powerful tool for classification tasks due to their ability to outperform traditional methods. Nevertheless, their effectiveness relies heavily on the availability of large, varied, and labeled datasets, which are often not available. To counter this constraint, data augmentation techniques have emerged, leveraging existing data to generate additional, variant data. Extending these techniques to multi-dimensional time series data, such as the transportation mode detection data considered in this thesis, however, introduces challenges. In response, generative models such as Variational Autoencoders (VAEs) have shown promising advancements.

In this context, this thesis investigates the application of the Iterative Hierarchical Data Augmentation (IHDA) algorithm for ANNs, which represents a VAE-based data augmentation technique. The IHDA method utilizes VAEs not only to generate new data samples but also to map existing data to a lower-dimensional latent space, which is then utilized for identifying samples that might require additional training. The proponents of this method, Khan and Fraz, reported an accuracy elevation for the considered transportation mode detection classifier from 83% to 92%. However, due to the absence of publicly accessible code for this algorithm, the initial step of this thesis involved implementing the IHDA algorithm. Further, this research proposed and incorporated advancements like the σ-VAE, aimed to improve the generative capacity of the VAE and refining its latent space mapping. Additionally, the Kullback-Leibler (KL) divergence was introduced as a similarity metric, aiming to optimize the identification process of samples that require retraining.

Unfortunately, the results reported by Khan and Fraz could not be reproduced in this study. Furthermore, despite the potential shown by the σ-VAE to improve the generative capacity and refine the latent space mapping, along with the enhanced sample identification through the KL divergence, these enhancements did not lead to an overall improvement in the IHDA algorithm. This was primarily attributed to the low generative performance of the VAEs utilized, which also hindered a thorough evaluation of the effectiveness of the IHDA algorithm.

Given these outcomes, it is suggested that future work should focus on employing more complex VAE models with the potential to enhance their generative performance, which, in turn, could improve the IHDA algorithm's overall effectiveness.

Table of Contents

List of Figures

List of Tables

Preface

First and foremost, I extend my heartfelt gratitude to my supervisors: Dr. Manon Kok from the Department of Systems and Control at TU Delft, Dr. Qinrui Tang, Kanwal Jahan, and Dr. Michael Roth from DLR. Their consistent support, invaluable guidance, and constructive feedback throughout the course of our regular progress meetings have significantly shaped the direction and depth of this thesis.

I am grateful to these individuals for giving me the unique opportunity to conduct my graduation project at DLR. This experience has been an essential part of my academic journey, promoting personal growth and deepening my understanding of my field. Therefore, I hope that the collaboration between TU Delft and DLR will continue, providing similar opportunities for future students.

As I mark the end of my academic tenure, I look back with gratitude at my time spent at TU Delft. This institution has equipped me with a solid academic foundation while fostering an environment that encouraged exploration and growth. I will always value the support and guidance I received throughout my journey, and take these memories with me into the next phase of my professional life.

Chapter 1

Introduction

1-1 Transportation Mode Detection and Its Application

In 2021, more than 80% of the global population owned a smartphone, which is expected to increase in the future [1]. This statistic, coupled with the fact that the average smartphone user spends three hours and fifteen minutes on their device each day, illustrates that smartphones significantly influence our daily lives [2]. In response to this trend, smartphone manufacturers and application developers continually strive to improve the smartphones' ability to understand the context of their users in order to provide them with the best possible and personalized service.

As commuting consumes a significant portion of an individual's daily routine, with an average of up to 80 minutes spent on traveling [31], the research field of transportation mode detection (TMD) has emerged in recent years. It focuses on using body-worn sensors, such as those embedded in smartphones, to identify the transportation means used by an individual, such as walking, biking, or traveling by car.

TMD algorithms can find applications across a wide range of domains. For instance, modern smartphones already provide fitness applications allowing users to track their time spent with activities such as walking or running [3]. By providing feedback on the amount of physical activity performed, smartphones can assist users in adopting a healthier lifestyle. Additionally, TMD algorithms can also be leveraged in urban transportation planning, where understanding travel demands and identifying modes of transportation is crucial. Traditional methods of gathering this information, such as questionnaires, travel diaries, or telephone interviews, are often expensive, limited to a specific area, and prone to human-induced errors [38]. TMD algorithms offer a more cost-effective and accurate alternative, mitigating these challenges and providing valuable insights into transportation patterns and needs [48, 49]. Another notable application of TMD algorithms is user profiling. By analyzing an individual's travel behavior, TMD algorithms enable targeted advertising, personalized recommendations, and the delivery of tailored content, thereby improving user experiences [18]. These examples highlight the versatility and significance of TMD algorithms in various fields and emphasize the need for further research and development to unlock their full potential.

1-2 The Sussex-Huawei Locomotion Challenge

The research of TMD dates back to 2006 when the first studies were published [36, 50]. Early research faced a significant challenge, as each research group had to create its own dataset, which was both time-consuming and costly. In addition, the datasets varied in terms of parameters such as the recording length, the smartphone's position on the body, the number of users, and the mode of transportation. Consequently, comparing and drawing conclusions from different research results was difficult. Thus, a publicly available baseline dataset was required that could be used by different research groups and several have been created since [4, 10].

At the time of writing and to the best of the authors' knowledge, the largest and latest published dataset in the research community is the Sussex-Huawei Locomotion (SHL) dataset. It contains 2812h of labeled data collected by three participants. Each participant carried four smartphones at the body locations: *Hand, Torso, Hip Pocket* and *Backpack*. Furthermore, the dataset distinguishes between eight transportation modes: *Still, Walk, Run, Bike, Car, Bus, Train* and *Subway* [14].

The SHL dataset is part of the Sussex-Huawei Locomotion (SHL) Challenge, a research competition jointly organized by Huawei and the University of Sussex to encourage research in the field of TMD. Between 2018 and 2021, the SHL Challenge was held annually and attracted a total of 63 participating teams [41, 43, 44, 45]. Each year, a new subset of the SHL dataset was made publicly available, and teams were invited to compete against one another by developing classifiers based on the latest SHL subset. As a result, recent TMD research has predominantly focused on using the SHL dataset.

1-3 Motivation

TMD involves utilizing data from body-worn sensors to identify a user's mode of transportation. The mathematical objective is to find the relationship between the sensor data and the transportation mode. This is typically accomplished using Machine Learning (ML) algorithms, which build mathematical models from sample data to make predictions or classifications without being explicitly programmed to do so.
Deep Learning (DL) models are based on Artificial Neural Network (ANN) and represent a subclass of ML models. However, for the purpose of this thesis, DL will be considered as a separate class of models. Therefore, a distinction is made between ML and DL models, with the latter being based on ANN.

Deep Learning models are powerful models that have proven to perform well across a wide range of learning tasks and applications, such as computer vision [17, 26] and natural language processing [8, 39]. According to the results of the four SHL Challenges, Deep Learning models are also capable of performing well for classification in TMD applications. In three out of the four challenges, the best DL model surpassed the best ML model in terms of classification performance.

In the 2019 SHL challenge, which was the only competition in which the best ML approach outperformed the best DL approach, the training and testing data were collected with smartphones placed at different positions on the body, with the testing data having a distinct body position compared to the training data [43]. This resulted in a significant mismatch between the two datasets. Additionally, there was a noticeable drop in classification performance compared to its previous year's challenge, SHL 2018, in which data from a single, consistent body position was used for both training and testing [41].

These observations suggest that the mismatch between training and testing data might affect classification performance. To provide context, the concept of *overfitting* should be introduced, which occurs when a model is trained to fit training data too closely, resulting in poor performance on new data, particularly when collected differently [16]. Due to their high modeling power, DL models are particularly prone to it. Consequently, the performance of DL models relies on the availability of large datasets with a high amount of variation. However, generating labeled data can be time-consuming and costly, which often limits the availability of large datasets.

A technique known as Data Augmentation (DA) can be employed to increase the amount of available data by applying various transformations to existing data samples to generate new, slightly different samples. Recent advancements in generative models, such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs), have made them particularly appealing for this purpose. These models have shown promising results when applied to various types of data across domains, including medical image augmentation and acoustic modeling [5, 11, 30]. Despite these successes, they have not received significant attention in the field of TMD. Consequently, there is a need to explore their potential applications and effectiveness in this context.

Compared to GANs, VAEs come with the advantage that they can also be used for dimensionality reduction of data samples [15, 24]. This process involves transforming high-dimensional data into a lower-dimensional representation while preserving essential features, which leads to a more compact and efficient representation. This dual functionality of VAEs allows them to generate artificial samples and simplify the representation of their input data.

Taking advantage of this property, Khan and Fraz [22] proposed a VAE-based data augmentation technique that identifies data samples within the training dataset that could benefit from additional sampling. Using a subset of the SHL dataset, they reported an improvement in the classification accuracy of their TMD classifier from 83% to 92%, which is a promising result. Nevertheless, the absence of the necessary hyperparameters and code associated with the proposed technique creates a gap in understanding and replicating the Iterative Hierarchical Data Augmentation (IHDA) algorithm. This situation underscores the necessity for further exploration.

1-4 Purpose of this Thesis

Given these limitations, there is a clear path leading to the main purpose of this thesis. This thesis is motivated by the compelling findings presented by Khan and Fraz [22], which highlight the potential of the IHDA algorithm. However, to address existing gaps caused by missing hyperparameters and unavailable code, the first step of this thesis is to reimplement the IHDA algorithm as initially proposed.

This task holds significant importance, not only for gaining insights into the operational mechanics of the IHDA algorithm but also for directly benchmarking any proposed modifications against the original research.

Following a successful reimplementation, the thesis conducts a comprehensive evaluation of the IHDA algorithm's performance, specifically in its ability to enhance a TMD classifier using the SHL dataset. The aim here goes beyond merely validating the improvements reported by Khan and Fraz [22], as the goal is to uncover potential areas for further refinement.

These initial investigations set the stage for the objective of this thesis: the identification, implementation, exploration, and assessment of potential enhancements to the IHDA algorithm. The aim is to ascertain whether these enhancements can indeed improve the performance of the IHDA algorithm beyond the achievements reported in the original study by Khan and Fraz [22].

1-5 Organization

After having laid out the motivation and purpose of this thesis, the subsequent sections will detail the structure and organization of the remainder of this work. Chapter 2 introduces the fundamental concept of ANN. It outlines the principles of neural networks, details their operation, and describes the procedure for their training. This foundational knowledge is required for facilitating an in-depth understanding of an ANN architecture named VAE, which is the focus of Chapter 3.

Chapter 3 explores the Variational Autoencoder (VAE), including an examination of its various variations, such as β-VAE and σ-VAE. This chapter delivers the essential groundwork, enabling the reader to thoroughly understand the context and content of the subsequent chapter.

Transitioning to Chapter 4, the focus pivots towards the practical application of VAEs for Data Augmentation. A specific emphasis is placed on the IHDA algorithm, which presents the VAE-based data augmentation technique being studied in this thesis.

Chapter 5 outlines the methodologies employed in this thesis and presents the proposed enhancements to the IHDA algorithm. It prepares the reader for the experimental phase of the thesis by providing detailed justifications and explanations for the suggested improvements to the IHDA algorithm.

Chapter 6 presents the findings derived from the experiments conducted and provides an analysis of the compiled results. The individual results from the different experiments are further discussed in relation to each other in Chapter 7.

Chapter 7, the final chapter, undertakes an examination of the results, drawing connections between the outcomes of the various experiments to formulate an understanding of the research. Additionally, this chapter outlines potential directions for future research, identifying promising fields for further study and continued investigation.

Chapter 2

Fundamentals of Machine Learning and Artificial Neural Networks

This chapter is dedicated to exploring the core concepts of Machine Learning (ML), Deep Learning (DL), and Artificial Neural Networks (ANNs) as utilized in this thesis. Its objective is to provide a comprehensive understanding by introducing utilized terminology and explaining the design workflow for these models. Additionally, the chapter explores the model training process, with a specific emphasis on backpropagation-based gradient descent. This technique will be explained in the context of its application to dense layers. However, as convolutional layers are also employed in this thesis as a secondary layer architecture, the chapter concludes with a brief overview of their fundamental principles and relevant terminology.

2-1 Introduction to Machine Learning: Understanding the Workflow and Terminology

In transportation mode detection (TMD), ML and DL models are utilized for the task of classification, which implies they are used to approximate the mapping between sensor data-based inputs and their corresponding transportation modes. A concise overview of the workflow followed to develop these models will be presented to aid the reader's comprehension of the ML and DL terminology employed in this thesis. The ML workflow, illustrated in Figure 2-1, serves as a representation of the development process for both ML and DL models. For a more in-depth understanding of this workflow and the associated terminology, detailed explanations can be found in [6, 16].

The ML workflow begins with the acquisition of a raw dataset. This step involves collecting a broad set of data that is relevant to the model being developed. If the data is labeled, it means that, in the case of TMD, for the collected sensor data, the corresponding transportation mode is explicitly specified and thus available during the development process of the ML and DL models. As part of the data acquisition process, the raw data is prepared to make it suitable for subsequent steps in the workflow. A common preparation step is removing any incorrect data or outliers that could adversely affect the quality of the dataset.

Once this is done, the prepared dataset is used for the steps of feature extraction and feature engineering. A feature is a measurable property that serves as an input to a ML model. Accordingly, ML and DL models make predictions or classification decisions based on features. In feature extraction and feature engineering, domain knowledge is used to select and transform the most relevant variables from the prepared dataset. The dataset resulting from the feature extraction and engineering step contains the features and will be used for training the model.

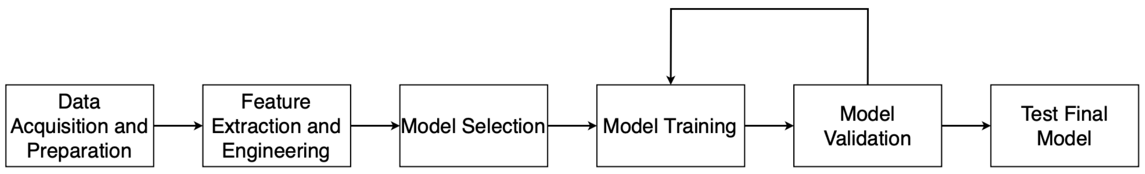
Data Acquisition and Preparation
Feature Extraction and Engineering
Model Selection
Model Training
Model Validation
Test Final Model

Figure 2-1: Flowchart depicting the process of developing machine learning (ML) and deep learning (DL) models, referred to as the machine learning workflow.

Selecting an appropriate model type and architecture is crucial to the model selection process. Numerous architectural approaches are available, each with its own strengths and weaknesses. Therefore, the model selection step should be guided by choosing an architecture most suitable for the model's intended purpose.

In the final steps of the ML workflow, the dataset is split into three non-overlapping subsets: the training dataset, the validation dataset, and the testing dataset.

The mapping of an ML or DL model is defined by its model parameters. These parameters need to be adjusted to allow the model to accurately represent the desired input-output mapping, such as the mapping from features to labels in the case of classification tasks. To automate the process of adjusting the model parameters, a chosen training algorithm is utilized. The training dataset serves as a guide to the training algorithm, instructing it on how to effectively update the model parameters.

After the training algorithm has utilized the complete training dataset once to adjust the model parameters, the model is then validated against the validation dataset. This dataset serves to assess the model's ability to generalize its learning from the training dataset to new, unseen data.

The process of training and validation is typically repeated for a defined number of iterations, called epochs. Monitoring the model's performance on both the training and validation dataset allows for detecting overfitting. Overfitting is indicated by an increasing performance of the model on the training dataset while the performance on the validation dataset remains constant or even decreases, which suggests that the model is memorizing the training data rather than effectively learning to generalize to new data.

Upon completion of the training and validation processes, the final model is evaluated using the testing dataset. This dataset consists of data that the model has never encountered before, providing an unbiased assessment of the model's performance. It serves as an indicator of how the model will likely perform when presented with entirely new data, such as during real-world application. Furthermore, it informs the decision of whether to retain the current model or restart the ML workflow at the point where improvements are believed to be possible.

As DL models are utilized during this thesis, the following sections will provide a more detailed exploration of their architecture and training process.

2-2 Introduction to Artificial Neural Networks

Deep Learning (DL) models are based on Artificial Neural Network (ANN), which draw inspiration from the structure and function of biological neural networks in the brain. ANNs consist of interconnected nodes, known as artificial neurons, that process and transmit information through weighted connections, emulating the communication between neurons in the brain through electrical and chemical signals [6, 16].

As artificial neurons serve as the fundamental building blocks of every ANN, it is essential to understand their structure and operation. Therefore, the following section aims to provide a detailed explanation of artificial neurons, enabling a better understanding of the fundamental principles behind ANNs, which will be further discussed in subsequent sections.

2-2-1 Introduction to Artificial Neurons

The artificial neuron represents a function that takes n inputs, multiplies every input x_i by a weight w_i, and adds a bias b to the resulting product [6, 16]. The sum of the weighted inputs and bias is called the activation a which is mapped to the artificial neuron's output o by an activation function $g()$. By choosing the activation function $g()$ to be non-linear, non-linearity is introduced to the mapping. This process can be schematically represented as shown in Figure 2-2, and mathematically denoted as

$$\begin{aligned} a &= \sum_{i=1}^{n} x_i w_i + b, \\ o &= g(a). \end{aligned} \tag{2-1}$$

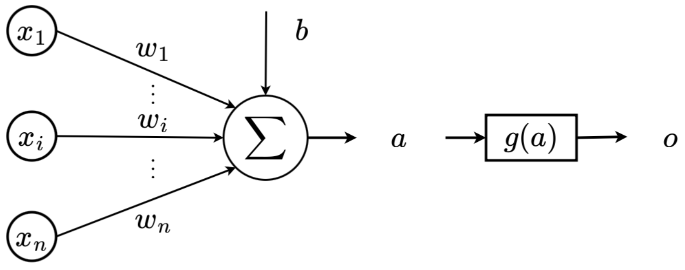

Figure 2-2: Schematic representation of an artificial neuron, with inputs x, weights w, bias b, activation a, activation function $g(a)$ and resulting output o.

2-2-2 Basic Architecture of Artificial Neural Networks

An Artificial Neural Network (ANN) is created by combining two or more artificial neurons, which results in a mapping composed of the mappings implemented by each neuron within the network [6, 16]. For this reason, the choice of a nonlinear activation function $g()$ for the artificial neurons results in a nonlinear mapping of the ANN. In fact, it enables ANNs to approximate any computable mapping to an arbitrary precision [46]. This makes them suitable for modeling complex real-world relations, such as the linkage between sensor data and the transportation mode. However, the precision at which an ANN can approximate a mapping is limited by the number of artificial neurons in the network.

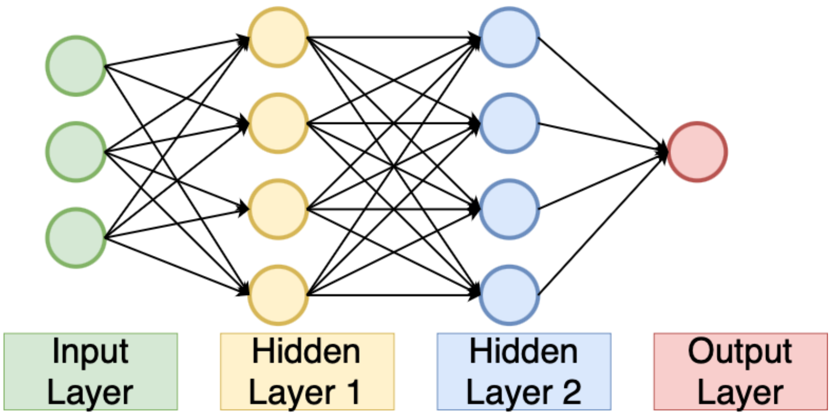

Figure 2-3: Schematic representation of a Dense Neural Network (DNN) composed of an input layer, a hidden layer, and an output layer.

Artificial neurons in ANNs are organized into layers, including an input layer that receives the initial input $\mathbf{x} \in \mathbb{R}^N$, one or more hidden layers that perform computations on that input, and an output layer producing the final output of the ANN [6, 16]. Each layer receives its input from the previous layer and provides output to the next layer, as illustrated in Figure 2-3.

To describe how the input $\mathbf{x} \in \mathbb{R}^N$ of an ANN is transformed in the different layers of the network, the notation listed in Table 2-1 will be used.

Notation	Description
r_l	Number of neurons in the l-th layer
w_{ij}^l	Weight associated with the j-th input to the i-th neuron in the l-th layer
b_i^l	Bias term for the i-th neuron in the l-th layer
a_i^l	Activation for the i-th neuron in the l-th layer
o_i^l	Output of the i-th neuron in the l-th layer
$g_l()$	Activation function for the neurons in the l-th hidden layer
$g_o()$	Activation function for the neurons in the output layer

Table 2-1: Notations used to describe the transformation of inputs in the layers of an Artificial Neural Network (ANN).

The input layer consists of N input neurons, each representing one input to the ANN. In contrast to artificial neurons, there is no processing of information involved in the input neurons, which implies that no mathematical operations are applied [6, 16]. Instead, they pass the inputs to the following hidden layer. Consequently, the output of the input layer, denoted as $\mathbf{o}^0 \in \mathbb{R}^N$, is identical to its input which is expressed as

$$\mathbf{o}^0 = \mathbf{x}. \tag{2-2}$$

The l-th hidden layer, which is composed of r_l number of artificial neurons, receives as input the output $\mathbf{o}^{l-1} \in \mathbb{R}^{r_{l-1}}$ from the previous layer, which could be either the input layer or the preceding hidden layer. The output of the hidden layer is then transmitted to the succeeding hidden layer or directly to the output layer [6, 16].
The arrangement of the artificial neurons in the hidden layers determines how the input $\mathbf{x}$ of the ANN are transformed and how information is processed in the network. Different architectural approaches are available due to the versatility of combining artificial neurons in various ways. In this thesis, both dense layers and convolutional layers will be utilized. However, for the purpose of simplicity, the subsequent sections will focus on explaining the training of neural networks using an example of ANNs comprised of dense layers, known as a Dense Neural Network (DNN).

A dense layer comprises multiple artificial neurons, where all the previous layer's outputs are passed as inputs to each neuron of the dense layer, as depicted in Figure 2-3. The transformation of a dense hidden layer will be denoted as

$$\begin{aligned} \mathbf{a}^l &= \mathbf{w}^l \mathbf{o}^{l-1} + \mathbf{b}^l, \\ \mathbf{o}^l &= g_l(\mathbf{a}^l). \end{aligned} \tag{2-3}$$

Here, $\mathbf{w}^l \in \mathbb{R}^{r_l \times r_{l-1}}$ and $\mathbf{b}^l \in \mathbb{R}^{r_l}$ denote the weights and biases of the l-th layer, respectively. Moreover, $\mathbf{a}^l \in \mathbb{R}^{r_l}$ and $\mathbf{o}^l \in \mathbb{R}^{r_l}$ represent the activations and subsequent outputs of the l-th layer, whereas $\mathbf{o}^{l-1} \in \mathbb{R}^{r_{l-1}}$ captures the outputs from the preceding $(l-1)$-th layer.

The output layer processes the output of the last hidden layer $\mathbf{o}^{L-1} \in \mathbb{R}^{r_{L-1}}$ by using one or multiple artificial neurons, producing the output $\mathbf{o}^L \in \mathbb{R}^{r_L}$ of the resulting ANN [6, 16]. It is common to use the same type of activation function $g_l()$ for the artificial neurons of different hidden layers. In contrast, depending on the purpose of the network, a different activation function $g_o()$ might be utilized for the artificial neurons of the output layer. Therefore, similar to the dense layer's input-output transformation provided by (2-3), the output layer's input-output transformation in a DNN with L layers can be expressed as

$$\begin{aligned} \mathbf{a}^L &= \mathbf{w}^L \mathbf{o}^{L-1} + \mathbf{b}^L, \\ \mathbf{o}^L &= g_o(\mathbf{a}^L), \end{aligned} \tag{2-4}$$

where $g_o()$ denotes the output activation function, introduced to distinguish it from the activation functions used in the hidden layers.

As randomly initializing the weights and biases $\{\mathbf{w}^l, \mathbf{b}^l\}_{l=1}^{L}$ of an ANN is unlikely to yield the desired input-output mapping, these model parameters require optimization, a process that will be discussed in the subsequent section.

2-3 Training Dense Neural Networks

Training an ANN refers to the process of identifying a set of network weights and biases $\{\mathbf{w}^l, \mathbf{b}^l\}_{l=1}^{L}$, collectively referred to as network parameters, that result in an accurate approximation of the desired mapping [6, 16]. To accomplish this, a training data set $\mathbf{X} = \{\mathbf{x}_k, y_k\}_{k=1}^{K}$, consisting of K input-output pairs $\{\mathbf{x}_k, y_k\}$, is used to adjust the network's weights and biases iteratively to increase its performance on a given task. To provide a clearer understanding of this training process, the following section will describe the training of a DNN with $L - 1$ hidden layers, using a method known as Gradient Descent (GD).

GD refers to an iterative optimization algorithm for finding a local minimum of a differentiable function. In the context of training an ANN, this indicates that the first step involves defining a differentiable loss function dependent on the network's weights and biases.

2-3-1 Loss Function

The task of selecting an appropriate loss function for an ANN goes beyond simple dependency on the network's weights and biases $\{\mathbf{w}^l, \mathbf{b}^l\}_{l=1}^{L}$. The chosen function should also ensure that deviations from the ANN's objective result in an increasing loss value. Consequently, by adjusting the model parameters to minimize the loss function, the ANN improves its ability to achieve its objective. An example of a suitable loss function for measuring the difference between the true output value y_k and the network's output is the Mean Square Error (MSE) [6, 16]. The network's output for a given training data sample input $\mathbf{x}_k$ is denoted as $\mathbf{o}^L(\mathbf{x}_k)$, and the MSE is defined as

$$\mathcal{L}(\mathbf{w}, \mathbf{b}) = \frac{1}{K} \sum_{k=1}^{K} (y_k - \mathbf{o}^L(\mathbf{x}_k))^2. \tag{2-5}$$

In this equation, $\mathcal{L}(\mathbf{w}, \mathbf{b})$ denotes the loss function, which quantifies the discrepancy between the true output values $y_k \in \mathbb{R}^{r_L}$ and the network's predicted outputs $\mathbf{o}^L(\mathbf{x}_k) \in \mathbb{R}^{r_L}$. The summation is taken over all K training examples. Accordingly, minimizing the MSE corresponds to reducing the average squared difference between the predicted outputs and the true outputs, aligning the network's predictions with the desired targets.

2-3-2 Gradient Descent

Once an appropriate loss function $\mathcal{L}(\mathbf{w}, \mathbf{b})$ has been defined, an algorithm is required to find a set of model parameters $\{\mathbf{w}^l, \mathbf{b}^l\}_{l=1}^{L}$ that minimize this function, with one of these algorithms being Gradient Descent (GD).

The rationale behind discussing Gradient Descent here is primarily because many other common training methodologies, such as RMSprop, Adam, or AdaGrad, are variations of GD [33]. Consequently, understanding GD provides sufficient background information to grasp the basics of these other training techniques in the context of this thesis.

In this section, for the purpose of simplicity, the model parameters $\{\mathbf{w}^l, \mathbf{b}^l\}_{l=1}^L$ will be denoted as $\mathbf{v} \in \mathbb{R}^H$, where H represents the dimension of the parameters.

The gradient of the loss function $\mathcal{L}(\mathbf{v})$ with respect to the model parameters will be denoted as $\nabla\mathcal{L}(\mathbf{v})$, and it characterizes a local or global minimum of the function when it vanishes

$$\nabla\mathcal{L}(\mathbf{v}) = 0. \tag{2-6}$$

Here, $\nabla\mathcal{L}(\mathbf{v})$ represents the partial derivatives of $\mathcal{L}(\mathbf{v})$ with respect to model parameter contained in $\mathbf{v}$ and can be expressed as

$$\nabla\mathcal{L}(\mathbf{v}) = \Big(\frac{\partial\mathcal{L}(\mathbf{v})}{\partial v_1}, \frac{\partial\mathcal{L}(\mathbf{v})}{\partial v_2}, \cdots, \frac{\partial\mathcal{L}(\mathbf{v})}{\partial v_H}\Big)^T. \tag{2-7}$$

The loss function $\mathcal{L}(\mathbf{v})$ is typically highly nonlinear and dependent on the model parameters $\mathbf{v}$, making it impossible to calculate an analytical solution to (2-6). Therefore, an alternative approach is required to minimize the loss function $\mathcal{L}(\mathbf{v})$, such as GD. It involves selecting some initial values $\mathbf{v}_0$ for the model parameters before iteratively moving through the parameter space until a minimum is reached [6, 16]. The network parameters are updated according to the following rule

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \Delta\mathbf{v}_i, \tag{2-8}$$

where the subscript i indicates the model parameters after the i-th parameter update step. Moreover, the values used to update the model parameters are contained in $\Delta\mathbf{v}_i$, which can be denoted as

$$\Delta\mathbf{v}_i = (\Delta v_1, \Delta v_2, \cdots, \Delta v_H)_i^T. \tag{2-9}$$

Gradient Descent updates the network parameters by using the negative gradient $-\nabla\mathcal{L}(\mathbf{v}_i)$ as part of the update vector $\Delta\mathbf{v}_i$, resulting in the following step for updating the parameters

$$\mathbf{v}_{i+1} = \mathbf{v}_i + \Delta\mathbf{v}_i = \mathbf{v}_i - \eta\nabla\mathcal{L}(\mathbf{v}_i). \tag{2-10}$$

The gradient $\nabla\mathcal{L}(\mathbf{v})$ represents the direction in which the loss function $\mathcal{L}(\mathbf{v})$ increases the most, while the GD algorithm updates the model parameters in the opposite direction, where the loss function decreases the most. Further, the hyperparameter $\eta > 0$ in Equation (2-10) is commonly referred to as the learning rate, and it determines the extent to which the model parameters are adjusted in the direction of the negative gradient [6, 16].

The effectiveness of the GD algorithm can be demonstrated based on the local linear approximation of the loss function $\mathcal{L}(\mathbf{v})$. Under this assumption, a sufficient small step $\Delta\mathbf{v}$ taken in the parameters space results in a change of the loss function $\Delta\mathcal{L}(\mathbf{v})$ approximately equal to

$$\Delta\mathcal{L}(\mathbf{v}) \simeq \nabla\mathcal{L}(\mathbf{v})\Delta\mathbf{v} = \sum_{h=1}^{H} \frac{\partial\mathcal{L}(\mathbf{v})}{\partial v_h}\Delta v_h. \tag{2-11}$$

By using Equation (2-11), it can be shown that the parameter update vector $\Delta \mathbf{v} = -\eta \nabla \mathcal{L}(\mathbf{v})$ used by the GD algorithm results in a negative change of the loss function $\Delta \mathcal{L}(\mathbf{v})$, hence the loss function value has decreased after updating the model parameters

$$\Delta \mathcal{L}(\mathbf{v}) \simeq \nabla \mathcal{L}(\mathbf{v}) \Delta \mathbf{v} = -\eta \nabla \mathcal{L}(\mathbf{v}) \nabla \mathcal{L}(\mathbf{v}) = -\eta ||\nabla \mathcal{L}(\mathbf{v})||^2 \leq 0. \tag{2-12}$$

However, Equation (2-12) is based on local linear approximation of the loss function $\mathcal{L}(\mathbf{v})$, which does not hold if the parameter update step $\Delta \mathbf{v}$ is taken too large. Since the size of the update step can be controlled by the learning rate η, it needs to be chosen carefully.

If the learning rate is too high, the algorithm can overshoot the minimum and diverge, resulting in failure to converge to a minimum of the loss function $\mathcal{L}(\mathbf{v})$. On the other hand, if the learning rate is too low, the algorithm may converge very slowly or get stuck in a suboptimal solution. Therefore, choosing an appropriate learning rate is a critical task and often involves a trial-and-error process to find the optimal learning rate that balances convergence speed and stability.

Finally, the parameter update rule of the GD algorithm, specified by Equation (2-10), can be translated into an update rule for the weights $\mathbf{w}^l$ and biases $\mathbf{b}^l$ defining layer l of a DNN as follows

$$\mathbf{w}^l_{i+1} = \mathbf{w}^l_i - \eta \frac{\partial \mathcal{L}(\mathbf{w}^l_i, \mathbf{b}^l_i)}{\partial \mathbf{w}^l_i} \quad \text{and} \quad \mathbf{b}^l_{i+1} = \mathbf{b}^l_i - \eta \frac{\partial \mathcal{L}(\mathbf{w}^l_i, \mathbf{b}^l_i)}{\partial \mathbf{b}^l_i}. \tag{2-13}$$

Stochastic Gradient Descent for Dense Neural Networks

By reconsidering the loss function introduced by (2-5), it is possible to interpret the total loss $\mathcal{L}(\mathbf{w}, \mathbf{b})$ as the average of the losses of each training input $\mathcal{L}_{\mathbf{x}_k}(\mathbf{w}, \mathbf{b})$. As a result, (2-5) can be rewritten as

$$\mathcal{L}(\mathbf{w}, \mathbf{b}) = \frac{1}{K} \sum_{k=1}^{K} (y_k - \mathbf{o}^L(\mathbf{x}_k))^2 = \frac{1}{K} \sum_{k=1}^{K} \mathcal{L}_{\mathbf{x}_k}(\mathbf{w}, \mathbf{b}). \tag{2-14}$$

Therefore, calculating the gradient $\nabla \mathcal{L}(\mathbf{w}, \mathbf{b})$ requires computing the average over the gradients of each training input $\nabla \mathcal{L}_{\mathbf{x}_k}(\mathbf{w}, \mathbf{b})$ represented as

$$\nabla \mathcal{L}(\mathbf{w}, \mathbf{b}) = \frac{1}{K} \sum_{k=1}^{K} \nabla \mathcal{L}_{\mathbf{x}_k}(\mathbf{w}, \mathbf{b}). \tag{2-15}$$

As a result, the larger the size of the training data set K, the more complex the gradient $\nabla \mathcal{L}(\mathbf{w}, \mathbf{b})$ calculation will be. Due to this, for large K it might become computationally infeasible to calculate the gradient for one iteration of the GD algorithm. This problem may be resolved by using a modified version of GD, known as Stochastic Gradient Descent (SGD) [6, 16]. Instead of utilizing the entire data set for the computation of the gradient, SGD selects a mini batch $\mathbf{S} = \{\mathbf{x}_s, y_s\}_{s=0}^{S}$ that is randomly drawn from the training set and approximates the gradient as

$$\nabla\mathcal{L}(\mathbf{w},\mathbf{b}) \approx \frac{1}{S}\sum_{s=1}^{S}\nabla\mathcal{L}_{\mathbf{x}_s}(\mathbf{w},\mathbf{b}). \tag{2-16}$$

Accordingly, the network weights $\mathbf{w}^l$ and biases $\mathbf{b}^l$ defining layer l are updated by SGD as follows

$$\mathbf{w}_{i+1}^l = \mathbf{w}_i^l - \frac{\eta}{S}\sum_{s=1}^{S}\frac{\partial\mathcal{L}_{x_s}(\mathbf{w}_t,\mathbf{b}_t)}{\partial\mathbf{w}_t} \quad \text{and} \quad \mathbf{b}_{t+1} = \mathbf{b}_t - \frac{\eta}{S}\sum_{s=1}^{S}\frac{\partial\mathcal{L}_{x_s}(\mathbf{w}_t,\mathbf{b}_t)}{\partial\mathbf{b}_t}. \tag{2-17}$$

Once the parameters have been updated, a new mini batch from the training data set is selected, and the procedure is repeated until all samples have been utilized once. It is then said that one training epoch has been completed.

2-3-3 Backpropagation

The presented GD and SGD algorithm requires the computation of the gradient $\nabla\mathcal{L}(\mathbf{w},\mathbf{b})$, and backpropagation represents an efficient method for doing so [6, 16]. Using the calculus chain rule, the gradient of the loss function with respect to the weight w_{ij}^l can be rewritten as

$$\frac{\partial\mathcal{L}(\mathbf{w},\mathbf{b})}{\partial w_{ij}^l} = \frac{\partial\mathcal{L}(\mathbf{w},\mathbf{b})}{\partial a_j^l}\frac{\partial a_j^l}{\partial w_{ij}^l} = \delta_j^l o_j^{l-1}. \tag{2-18}$$

The first term δ_j^l, called the error, can be interpreted as a measure of how sensitive the loss function is to changes in the activation a_j^l, whereas the second term corresponds to the output o_j^{l-1} of the previous layer's neuron j. Consequently, the gradient of the DNN can be calculated by knowing the errors and outputs for each neuron of the network. By utilizing the calculus chain rule, it is possible to determine how the error δ_j^{l+1} propagates from layer $l+1$ to its preceding layer l, given by

$$\delta_j^l = \frac{\partial\mathcal{L}}{\partial a_j^l} = \frac{\partial\mathcal{L}}{\partial a_j^{l+1}}\frac{\partial a_j^{l+1}}{\partial o_j^l}\frac{\partial o_j^l}{\partial a_j^l} = \delta_j^{l+1}w_j^{l+1}g'(a_j^l). \tag{2-19}$$

In matrix notation, this can be written as

$$\boldsymbol{\delta}^l = ((\mathbf{w}^{l+1})\boldsymbol{\delta}^{l+1}) \odot g_l'(\mathbf{a}^l) \quad \text{with} \quad \boldsymbol{\delta}^l = (\delta_1^l, \delta_2^l, \cdots, \delta_{p_l}^l)^T \tag{2-20}$$

where $\odot$ represents the Hadamard product.

As can be inferred from (2-20), computing the error terms of the last layer first enables the calculation of all preceding error terms in the network. Propagating the error through the ANN according to (2-20) is known as backpropagation of the network's error.
Considering the MSE loss function defined in (2-5) the error δ_j^L of the output layer L and node j can be calculated as

$$\delta_j^L = \frac{\partial \mathcal{L}(\mathbf{w}, \mathbf{b})}{\partial a_j^L} = \frac{2}{K} \sum_{k=1}^{K} (y_k - g_o(a_j^L(\mathbf{x}_k)) g_o'(a_j^L(\mathbf{x}_k)). \tag{2-21}$$

However, this expression depends on the activation a_j^L, which is influenced by the previous layer's output o_j^{L-1}. Thus, to calculate the error of the last layer and backpropagate it through the network, it is necessary to calculate first the output of all neurons. As a result, for every iteration of the GD algorithm using backpropagation, the algorithm proceeds as follows.

As a first step, the inputs of the network are propagated through the DNN, resulting in the outputs of all neurons. This step is referred to as the forward pass. In the following step, these outputs are used to calculate the errors associated with each weight by propagating the errors of the last layer backwards through the network in accordance with (2-20). This is referred to as the backward pass [6, 16]. Finally, having calculated all errors and neuron outputs of the network, the gradient with respect to each weight can be calculated according to (2-18). Further, the procedure to calculate the gradient with respect to the bias b_j^l using backpropagation is the same as the one discussed, with the exception that this gradient is defined as

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l. \tag{2-22}$$

2-4 Supplemental Elements of ANN Architecture

After furnishing a comprehensive outline of ANNs and their training processes, this section is dedicated to the introduction of two additional components of ANNs that will be referenced throughout this thesis. These are the convolutional layers and activation functions. Given their recurrence in the forthcoming sections of this thesis, the objective here is to equip the reader with essential background knowledge on these elements. For a more in-depth exploration, the reader is directed to [6, 16].

2-4-1 Convolutional Layers

The convolutional layer offers, next to the dense layer, an alternative layer architecture in the field of ANNs. The goal of this section is to introduce the fundamental principles of these convolutional layers and to familiarize the reader with the relevant terminology. Techniques such as backpropagation-based gradient descent, previously explored within the context of dense layers, can be extended to convolutional layers in a straightforward manner, and therefore will not be reiterated here.

The term "convolutional" derives from the mathematical convolution operation, which is described by

$$(\mathbf{x} * \mathbf{w})[n] = \sum_{m=-\infty}^{\infty} \mathbf{x}[n-m]\mathbf{w}[m], \tag{2-23}$$

where the $*$ denotes the convolution operator. The operation $(\mathbf{x} * \mathbf{w})[n]$ represents a weighted sum where each element of $\mathbf{x}$ is multiplied by a corresponding element in $\mathbf{w}$. Within the context of this operation, n marks a specific position within the sequence, and m is the variable sliding over all potential values. This sliding concept also manifests in convolutional layers where the kernel $\mathbf{w}$ slides across the layer's input $\mathbf{x}$.

In the context of convolution layers, both the input $\mathbf{x}$ and kernel $\mathbf{w}$ are finite in size, with the kernel's size needing specification during the network design phase. The kernel elements serve as the model parameters, and accordingly determine the layers input-output mapping. The values of these parameters are learned during training by applying algorithms like GD and backpropagation, as previously discussed in the context of training dense layers.

This sliding process, applied in each convolutional layer, moves the kernel $\mathbf{w}$ across the input sequence $\mathbf{x}$, capturing a kernel-sized part of the input data and convolving it with the kernel. This process effectively performs an element-wise multiplication of the kernel with the selected input values. Notably, convolutional layers can apply multiple kernels at once, where each kernel produces a corresponding output channel. An output channel fundamentally represents the result of convolving the input with a specific kernel.

Additionally, stride and padding are crucial parameters. The stride dictates the kernel's movement distance for each convolution operation, while padding involves adding extra elements, usually zeroes, to the input data's boundaries, aiding in managing the output's spatial dimensions. For instance, 'Same'-padding maintains the output's width and height identical to the original input by applying appropriate padding.

2-4-2 Activation Functions

As highlighted in Subsection 2-2-1 and 2-2-2, the activation function introduces non-linearity to the mapping of an artificial neuron, and accordingly to the mapping of an ANN. A variety of activation functions exists but throughout this thesis, only the Rectified Linear Unit (ReLU) and the Softmax activation will be utilized. Therefore, those two will be briefly discussed in this section.

The ReLU activation function is defined as

$$g(a) = \max(0, a), \tag{2-24}$$

and introduces non-linearity to the computations of individual artificial neurons by outputting zero for negative inputs and the input value itself for non-negative inputs.

On the other hand, the Softmax function operates more like an activation layer rather than a standalone activation function. It works collectively on the activations of all the r_l artificial neurons in a given layer l. The Softmax function takes the activations of the layer's neurons as input, applies the exponential function to each input element a_i, and then normalizes these values by dividing each one by the sum of all exponentials $\sum_{j=1}^{r_l} e^{a_j}$. This process generates output values for each neuron in the layer, which lie within the range of $[0, 1]$ and collectively sum to 1. Due to these characteristics, the Softmax function is widely employed as an output activation layer $g_o()$ in multiclass classification tasks, as it allows for the interpretation of each output as a probability that the input belongs to a specific class. The Softmax function is defined as

$$g(a_i) = \frac{e^{a_i}}{\sum_{j=1}^{r_l} e^{a_j}}, \tag{2-25}$$

where a_i symbolizes the i-th input element, while r_l represents the total number of input elements.

Chapter 3

Variational Autoencoders: Probabilistic Modeling and Latent Representations

The objective of this chapter is to provide a comprehensive understanding of Variational Autoencoders (VAEs), an architecture within Artificial Neural Networks (ANNs) that serves as the foundation for the data augmentation technique studied in this thesis. The VAE architecture is presented alongside its counterpart, the Autoencoder (AE), as they share similar structural characteristics. The chapter explores the workings of the VAE architecture, including the associated loss function known as the Evidence Lower Bound (ELBO), and discusses how backpropagation-based Gradient Descent (GD) algorithms can be applied to optimize this loss function.

Furthermore, two specific variants of the VAE, namely the σ-VAE and the β-VAE, are introduced, which hold relevance to this thesis. The chapter also addresses the phenomenon of posterior collapse, an important consideration when training VAEs.

3-1 Autoencoder

The Autoencoder (AE) represents an ANN architecture composed of two parts: an encoder $e(\mathbf{x}) : \mathbb{R}^N \mapsto \mathbb{R}^D$ and a decoder $d(\mathbf{z}) : \mathbb{R}^D \mapsto \mathbb{R}^N$. The encoder's objective is to encode its input data $\mathbf{x} \in \mathbb{R}^N$ in a representation called the latent vector $\mathbf{z} \in \mathbb{R}^D$, where each element of the latent vector is referred to as a latent variable. Further, this vector resides in a vector space, known as the latent space [6, 16].

Once this latent vector $\mathbf{z}$ has been obtained, the AE utilizes it to reconstruct its input data, resulting in its reconstructed input $\hat{\mathbf{x}} \in \mathbb{R}^N$. Thus, the AE maps $\mathbf{x} \mapsto \hat{\mathbf{x}}$ through the encoder $e(\mathbf{x}) \mapsto \mathbf{z}$ and the decoder $d(\mathbf{z}) \mapsto \hat{\mathbf{x}}$, as illustrated in Figure 3-1.

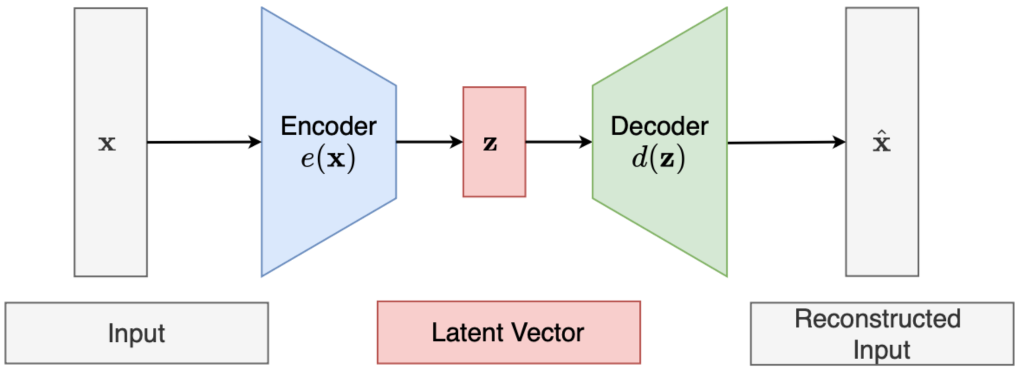

Figure 3-1: Schematic representation of an Autoencoder. The figure visually describes how the original input $\mathbf{x}$ is mapped to the latent vector $\mathbf{z}$ via the encoder $e(x)$, and then the input is reconstructed through the decoder $d(x)$ from $\mathbf{z}$ to $\hat{\mathbf{x}}$.

Training an AE refers to the process of learning the mapping $\mathbf{x} \mapsto \hat{\mathbf{x}}$. This is accomplished by adjusting the network weights, by algorithms such as the previously introduced GD algorithm, to optimize a loss function $\mathcal{L}_{AE}(\mathbf{x}, \hat{\mathbf{x}})$ that penalizes the difference between the original input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$. A commonly used loss function for training an AE is the Mean Square Error (MSE). Further, it's worth noting that the process of training an AE differs from classifier development, as it does not rely on the use of labels.

As a result of being able to perform the copying task $\mathbf{x} \mapsto \hat{\mathbf{x}}$, the AE may be able to extract meaningful features of the input data $\mathbf{x}$ in the latent vector $\mathbf{z}$ [6, 16]. If the dimension of the latent vector is selected to be smaller than that of the input data, the encoder is encouraged to extract and represent those meaningful features in a lower-dimensional latent space. Consequently, the latent vector $\mathbf{z}$ is usually of greater interest than the decoder output $\hat{\mathbf{x}}$, leading to the use of AEs for dimensionality reduction and feature extraction tasks.

Further insights into AE's dimensionality reduction and feature extraction capabilities can be gained by comparing it with the traditional representation learning technique, Principal Component Analysis (PCA). Under certain conditions, the encoding function of a single-layer Autoencoder with linear activation functions and MSE loss function is equivalent to PCA [29]. However, introducing non-linear activation functions in the hidden layers results in quite different characteristics, including the ability to convey other aspects of the input data. In this sense, the AE can be considered as a more powerful nonlinear generalization of PCA.

3-2 Variational Autoencoder

The understanding of the AE provides a basis for the study of VAEs, as both share a similar encoder-decoder architecture, making both of them applicable for feature extraction and dimensionality reduction. However, their formulations differ. Therefore, it may be beneficial to disregard the AE formulation in the context of this section.

3-2-1 The Evidence Lower Bound (ELBO)

Variational Autoencoders are probabilistic generative models that assume that the observed data sample $\mathbf{x} \in \mathbb{R}^N$ is generated by a stochastic process involving a stochastic latent vector $\mathbf{z} \in \mathbb{R}^D$ [24, 25].
The generative process can be subdivided into the following two steps: First, the latent vector $\mathbf{z}$ is generated by sampling it from a prior distribution $p_{\theta^*}(\mathbf{z})$. Then the resulting $\mathbf{z}$ is used to obtain $\mathbf{x}$ by sampling from the likelihood $p_{\theta^*}(\mathbf{x}|\mathbf{z})$. However, the values of the latent vector $\mathbf{z}$, as well as the true generative parameters θ^*, which define $p_{\theta^*}(\mathbf{z})$ and $p_{\theta^*}(\mathbf{x}|\mathbf{z})$, are unknown.

Therefore, ANNs are utilized to approximate the likelihood $p_{\theta^*}(\mathbf{x}|\mathbf{z})$ by a parametric family of distributions, referred to as the generative model $p_\theta(\mathbf{x}|\mathbf{z})$ or decoder. Furthermore, the true prior $p_{\theta^*}(\mathbf{z})$ is approximated by an isotropic centered Gaussian given by

$$p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathrm{I}). \tag{3-1}$$

Learning the stochastic mapping $p_\theta(\mathbf{x}|\mathbf{z})$ between the latent vector $\mathbf{z}$ and the observed data $\mathbf{x}$ requires an understanding of the vector $\mathbf{z}$ that generated the data $\mathbf{x}$. Thus, the posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is of interest, which can be expressed in terms of Bayes' theorem as

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})}. \tag{3-2}$$

Here, the evidence $p_\theta(\mathbf{x})$ of the observed data $\mathbf{x}$ taken as a function of the generative parameters θ can be attained by marginalizing the joint distribution $p_\theta(\mathbf{x}, \mathbf{z})$ over the latent vector $\mathbf{z}$ according to

$$p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z} = \int p_\theta(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z}. \tag{3-3}$$

This last equation requires exponential time to compute as it needs to be evaluated over all configurations of the latent space vector $\mathbf{z}$. As a result, the evidence $p_\theta(\mathbf{x})$ is regarded to be intractable, which implies that the posterior $p_\theta(\mathbf{z}|\mathbf{x})$, given by (3-2), is also intractable. Again, the solution proposed is to approximate it with a family of parametric distributions utilizing an ANN, which is referred to as the recognition model $q_\phi(\mathbf{z}|\mathbf{x})$ or encoder. To find recognition model parameters ϕ that result in a good approximation of the likelihood $p_\theta(\mathbf{z}|\mathbf{x})$, an evaluation metric is required. Therefore, the Kullback-Leibler (KL) is introduced, which is given by

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{x}, \mathbf{z})] + \log p_\theta(\mathbf{x}). \tag{3-4}$$

This non-negative measure quantifies the difference between two probability distributions and is zero if and only if both compared distributions are identical. Accordingly, it is aimed to find the generative and recognition model parameters $\hat{\theta}, \hat{\phi}$ that minimize the KL divergence between the likelihood $p_\theta(\mathbf{z}|\mathbf{x})$ and its approximation $q_\phi(\mathbf{z}|\mathbf{x})$, which is represented as

$$\hat{\theta}, \hat{\phi} = \underset{\theta,\phi}{\arg\min}\ D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})). \tag{3-5}$$

However, since the intractable evidence $p_\theta(\mathbf{x})$ appears in (3-4), it is impossible to compute the optimal model parameters $\hat{\theta}, \hat{\phi}$ directly [24, 25]. Therefore, the tractable measure called the ELBO, is introduced. It is analogous to the KL divergence up to a specific constant, hence to minimize $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$, one can study the ELBO instead.

For the derivation of the ELBO, a closer look is taken at the definition of the marginal log-likelihood $\log p_\theta(\mathbf{x})$. By multiplying and dividing by the posterior approximation $q_\phi(\mathbf{z}|\mathbf{x})$, this expression can be rewritten as

$$\begin{aligned} \log p_\theta(\mathbf{x}) &= \log \int p_\theta(\mathbf{x}, \mathbf{z}) dz = \log \int p_\theta(\mathbf{x}, \mathbf{z}) \frac{q_\phi(\mathbf{z}|\mathbf{x})}{q_\phi(\mathbf{z}|\mathbf{x})} dz \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \Big[\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})} \Big] \geq \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] = \text{ELBO}(\mathbf{x}) \end{aligned} \tag{3-6}$$

In the last step of (3-6), Jensen's inequality has been applied. Furthermore, it should be noted that the right hand side of the inequality corresponds to the ELBO$(\mathbf{x})$, providing a lower bound on the intractable log-likelihood of the evidence $p_\theta(\mathbf{x})$. That is why it is called Evidence Lower Bound (ELBO). Combining (3-4) and (3-6) indicates that the ELBO and the Kullback-Leibler divergence $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$ are related as

$$\begin{aligned} D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log q_\phi(\mathbf{z}|\mathbf{x}) - \log p_\theta(\mathbf{x}, \mathbf{z})] + \log p_\theta(\mathbf{x}) \\ &= -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] + \log p_\theta(\mathbf{x}) \\ &= -\text{ELBO}(\mathbf{x}) + \log p_\theta(\mathbf{x}). \end{aligned} \tag{3-7}$$

Inspecting (3-7) reveals that maximizing the tractable ELBO$(\mathbf{x})$ means either maximizing the intractable log-likelihood of the evidence $\log p_\theta(\mathbf{x})$ or minimizing KL divergence $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x}))$. Ideally, both are achieved. As a result of the first scenario, the generative model is improved $p_\theta(\mathbf{x}|\mathbf{z})$, whereas in the second scenario, the divergence between the posterior approximation $q_\phi(\mathbf{z}|\mathbf{x})$ and the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is minimized, resulting in a better recognition model $q_\phi(\mathbf{z}|\mathbf{x})$. Hence, maximizing the ELBO$(\mathbf{x})$ implies that, in the ideal case, the recognition ϕ and generative model parameters θ are optimized simultaneously [24, 25].

Rewriting the ELBO results in the final and tractable loss function $\mathcal{L}_{VAE}(\mathbf{x})$ used to train a VAE, given by

$$\begin{aligned} \text{ELBO}(\mathbf{x}) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}, \mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [\log p_\theta(\mathbf{x}|\mathbf{z})] + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} [p_\theta(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})} \Big[\log(p_\theta(\mathbf{x}|\mathbf{z})) \Big] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \\ &= -\mathcal{L}_{VAE}(\mathbf{x}). \end{aligned} \tag{3-8}$$

Note, that Equation (3-8) represents the ELBO evaluated on a single input data sample $\mathbf{x}$. However, when dealing with a dataset $\mathbf{X} = \{\mathbf{x}_k\}_{k=1}^{K}$, which comprises K input samples, the computation of the ELBO($\mathbf{X}$) extends to

$$\text{ELBO}(\mathbf{X}) = \sum_{k=0}^{K} \text{ELBO}(\mathbf{x}_k) = \sum_{k=0}^{K} \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x}_k)}\Big[\log(p_\theta(\mathbf{x}_k|\mathbf{z}))\Big] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}_k)||p_\theta(\mathbf{z})). \tag{3-9}$$

One important takeaway from this section is that compared to the AE, the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and decoder $p_\theta(\mathbf{x}|\mathbf{z})$ of a VAE both represent distributions. A Gaussian prior $p_\theta(\mathbf{z})$ for the latent space is commonly assumed, however, the specific assumptions for the encoder and the decoder distributions have not been introduced in this section. Thus, the following section will focus on a specific form of the VAE, called the strictly Gaussian VAE.

3-2-2 Strictly Gaussian Variational Autoencoder

In a strictly Gaussian VAE, in addition to assuming a Gaussian prior $p_\theta(\mathbf{z})$, the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and decoder $p_\theta(\mathbf{x}|\mathbf{z})$ both parameterize multivariate Gaussian distributions, characterized by their mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ [24, 25]. A visual representation of this concept is provided in Figure 3-2.

The three defining distributions of the strictly Gaussian VAE can be summarized as

$$\begin{aligned} p_\theta(\mathbf{z}) &= \mathcal{N}(\mathbf{0}, \mathbf{I}), \\ q_\phi(\mathbf{z}|\mathbf{x}) &= \mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x})), \\ p_\theta(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\Sigma}(\mathbf{z})), \end{aligned} \tag{3-10}$$

where $\boldsymbol{\mu}(\mathbf{x}) \in \mathbb{R}^D$ contains the mean values $\{\mu^d\}_{d=0}^{D}$ characterizing the distribution of each dimension of $\mathbf{z}$. Further, it is commonly assumed that the different dimensions of $\mathbf{z}$ are uncorrelated, which is why $\boldsymbol{\Sigma}(\mathbf{x}) \in \mathbb{R}^{D\times D}$ represents a diagonal matrix, containing the variance values $\{(\sigma^d)^2\}_{d=1}^{D}$ belonging to each dimension of $\mathbf{z}$. The same holds for the data sample $\mathbf{x}$ and its belonging mean $\boldsymbol{\mu}(\mathbf{z}) \in \mathbb{R}^N$ and its covariance matrix $\boldsymbol{\Sigma}(\mathbf{z}) \in \mathbb{R}^{N\times N}$.

Under the strictly Gaussian assumption, the KL divergence $D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}))$ term in the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$ for a data sample $\mathbf{x}$, as detailed in (3-8) can be expressed as

$$\begin{aligned} D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) &= D_{KL}(\mathcal{N}(\boldsymbol{\mu}(\mathbf{x}), \boldsymbol{\Sigma}(\mathbf{x}))||\mathcal{N}(\mathbf{0}, \mathbf{I})) \\ &= \frac{1}{2}\sum_{d=1}^{D}(\sigma^d)^2 + (\mu^d)^2 - 1 - 2\log\sigma^d. \end{aligned} \tag{3-11}$$

Further, the expectation term $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ of the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$ can be approximated by sampling a sufficiently large set of latent vectors $\{\mathbf{z}_s\}_{s=1}^{S}$ from the distribution $q_\phi(\mathbf{z}|\mathbf{x})$ [24]. Under this assumption the Monte Carlo estimate of the term $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})]$ becomes

$$
\begin{aligned}
\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\Sigma}(\mathbf{z}))] \\
&\approx \frac{1}{S}\sum_{s=1}^{S} \log \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}_s), \boldsymbol{\Sigma}(\mathbf{z}_s)) \\
&= \frac{1}{S}\sum_{s=1}^{S} -\frac{1}{2}(\log \det(\boldsymbol{\Sigma}(\mathbf{z}_s)) + (\mathbf{x}_s - \boldsymbol{\mu}(\mathbf{z}_s))^T(\boldsymbol{\Sigma}(\mathbf{z}_s))^{-1}(\mathbf{x}_s - \boldsymbol{\mu}(\mathbf{z}_s)) + N \log 2\pi).
\end{aligned}
\tag{3-12}
$$

Accordingly, the loss function $\mathcal{L}_{VAE}(\mathbf{x})$ for data sample $\mathbf{x}$ of a strictly Gaussian VAE is given by

$$
\begin{aligned}
\mathcal{L}_{VAE}(\mathbf{x}) &= -\,\mathrm{ELBO}(\mathbf{x}) \\
&= -\,\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log(p_\theta(\mathbf{x}|\mathbf{z}))\Big] + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})) \\
&\approx \frac{1}{S}\left[\sum_{s=1}^{S}\frac{1}{2}(\log \det(\boldsymbol{\Sigma}(\mathbf{z}_s)) + (\mathbf{x}_s - \boldsymbol{\mu}(\mathbf{z}_s))^T(\boldsymbol{\Sigma}(\mathbf{z}_s))^{-1}(\mathbf{x}_s - \boldsymbol{\mu}(\mathbf{z}_s)) + N \log 2\pi)\right] \\
&\quad + \frac{1}{2}\left[\sum_{d=1}^{D}(\sigma^d)^2 + (\mu^d)^2 - 1 - 2\log \sigma^d\right].
\end{aligned}
\tag{3-13}
$$

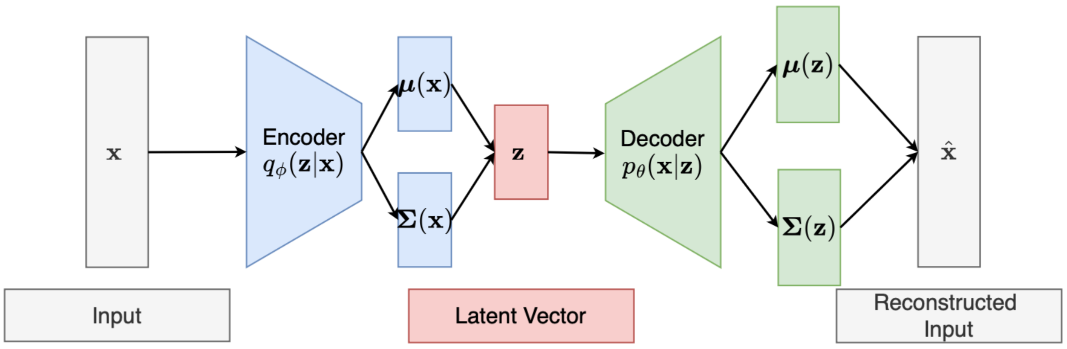
$\mathbf{x}$
Encoder
$q_\phi(\mathbf{z}|\mathbf{x})$
$\boldsymbol{\mu}(\mathbf{x})$
$\boldsymbol{\Sigma}(\mathbf{x})$
$\mathbf{z}$
Decoder
$p_\theta(\mathbf{x}|\mathbf{z})$
$\boldsymbol{\mu}(\mathbf{z})$
$\boldsymbol{\Sigma}(\mathbf{z})$
$\hat{\mathbf{x}}$
Input
Latent Vector
Reconstructed Input

Figure 3-2: Schematic representation of a Strictly Gaussian Variational Autoencoder (VAE). The figure illustrates how both the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ and the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ parameterize Gaussian distributions.

3-2-3 The Reparameterization Trick

Following the derivation of the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$, minimizing this function by using backpropagation-based algorithms for VAE training may seem intuitive. Nevertheless, a straightforward implementation of this is not achievable [24, 25]. As detailed in Section 2-3-3, the error must be propagated through the network, which necessitates calculating the gradient of the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$ with respect to the VAE parameters ϕ and θ. The VAE formulation, however, contains a stochastic sampling process of $\mathbf{z}$ over the latent space that is not differentiable. As a solution, the reparameterization trick is introduced, which extends the VAE input by an auxiliary random variable, $\boldsymbol{\epsilon} \in \mathbb{R}^D$. This allows to simulate the stochastic sampling process of $\mathbf{z}$ by computing it as

$$\mathbf{z} = \boldsymbol{\mu}(\mathbf{x}) + (\boldsymbol{\Sigma}(\mathbf{x}))^{0.5} \odot \boldsymbol{\epsilon} \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{3-14}$$

As a result, the reparameterization trick transformed the stochastic VAE model with input $\mathbf{x}$ into a deterministic model with inputs $\mathbf{x}$ and $\boldsymbol{\epsilon}$, allowing backpropagation-based algorithms to minimize the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$ [24, 25].

3-2-4 Sample Reconstruction Using a Strictly Gaussian VAE

Building on the reparameterization trick, the VAE framework maps the generated latent vector $\mathbf{z}$ back to a probability distribution $p_\theta(\mathbf{x}|\mathbf{z})$ in the input space using the decoder. From this distribution, the reconstructed input $\hat{\mathbf{x}}$ can be sampled. In the case where the decoder parameterizes a Gaussian distribution, as in the strictly Gaussian VAE framework, the sampling process for obtaining a reconstructed input can be described as

$$\hat{\mathbf{x}} = \boldsymbol{\mu}(\mathbf{z}) + (\boldsymbol{\Sigma}(\mathbf{z}))^{0.5} \odot \boldsymbol{\epsilon} \quad \text{with} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{3-15}$$

In practice, the covariance of the decoder $\boldsymbol{\Sigma}(\mathbf{z})$ is rarely used since adding $(\boldsymbol{\Sigma}(\mathbf{z}))^{0.5} \odot \boldsymbol{\epsilon}$ to the outer layer of the decoder simply introduces white noise to the reconstruction $\hat{\mathbf{x}}$ [9]. Then the decoder becomes a deterministic mapping that maps the latent variable to the artificially generated sample $\hat{\mathbf{x}}$ as

$$\hat{\mathbf{x}} = \boldsymbol{\mu}(\mathbf{z}). \tag{3-16}$$

3-3 The Importance of Selecting Decoder Variance in VAE

The common practice of disregarding the decoder covariance during the reconstruction phase often leads to neglecting the decoder covariance matrix throughout the entire development process of a VAE, by treating it as a constant. This practice involves neglecting the terms $\log \boldsymbol{\Sigma}(\mathbf{z})$ and $N \log(2\pi)$ in the VAE loss function, as they represent constants with zero derivatives and do not contribute to gradient calculations in optimization algorithms based on gradient descent [22]. Consequently, assuming a diagonal covariance matrix $\boldsymbol{\Sigma}(\mathbf{z})$ with fixed diagonal elements equal to 2, the VAE loss function for a strictly Gaussian VAE, represented by Equation (3-13), simplifies as

$$\mathcal{L}_{VAE}(\mathbf{x}) = N \cdot \mathrm{MSE}(\mathbf{x}_s, \boldsymbol{\mu}(\mathbf{z}s)) + \frac{1}{2}\left[\sum_{d=1}^{D}(\sigma^d)^2 + (\mu^d)^2 - 1 - 2\log\sigma^d\right]. \tag{3-17}$$

However, the practice of neglecting the decoder covariance matrix may lead to poor performing VAEs. For demonstration purposes, it is assumed that the decoder variance for the different dimensions of $\mathbf{x}$ are equal, resulting in the decoder covariance matrix $\boldsymbol{\Sigma}(\mathbf{z}) = (\sigma(\mathbf{z}))^2\mathbf{I}$. In this case, the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$ for a strictly Gaussian VAE, given by Equation (3-13), simplifies as

$$\begin{aligned}\mathcal{L}_{VAE}(\mathbf{x}) \approx & \frac{1}{S}\left[\sum_{s=1}^{S}\frac{1}{2}\Big(\log(\sigma(\mathbf{z}))^2 + \frac{N}{(\sigma(\mathbf{z}))^2}\mathrm{MSE}(\mathbf{x}_s, \boldsymbol{\mu}(\mathbf{z}_s)) + N\log 2\pi\Big)\right] \\ & + \frac{1}{2}\left[\sum_{d=1}^{D}(\sigma^d)^2 + (\mu^d)^2 - 1 - 2\log\sigma^d\right]\end{aligned} \tag{3-18}$$

From (3-18), it should be noted that the decoder variance acts as a weighting matrix between the MSE reconstruction loss and Kullback-Leibler divergence terms in the VAE-loss function. Accordingly, it determines how each of these terms contributes to the gradient calculation of the VAE loss function, which directly influences how the VAE model parameters are updated during the training phase with gradient-based optimization algorithms.

Therefore, while the decoder variance may not be used for reconstruction directly, assuming a fixed variance can result in one of the terms being overweight, preventing the global optimum of the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$ from being reached [34]. As a consequence, the lower bound on the marginal log-likelihood (ELBO($\mathbf{x}$)) of the data under consideration is not maximized, implying its marginal log-likelihood $\log p_\theta(\mathbf{x})$ may not be maximized as well. As a result, the VAE may not perform well, resulting in, for example, low-quality reconstructions [34].

Addressing this issue, the next section introduces two different VAE architectures that each offer a solution for the selection of the decoder covariance values. In the context of this thesis, both methods are discussed with regard to selecting a shared value for the diagonal elements of the decoder covariance matrix.

3-3-1 σ-VAE

The σ-VAE is a variant of the VAE that uses the VAE loss function $\mathcal{L}_{VAE}(\mathbf{x})$ to guide the selection of the decoder variance. According to Rybkin et al. [34], the VAE-loss function provides explicit instructions for selecting the optimal decoder variance: "the variance should be selected to minimize the weighted MSE loss while also minimizing the logarithm of the variance."

For the VAE loss function resulting from employing a single decoder variance value across all dimensions of the decoder output, as demonstrated in (3-18), an expression for the decoder variance that satisfies these conditions can be derived by differentiating the loss function and setting it equal to zero. This leads to the expression

$$(\sigma(\mathbf{z}))^2 = \arg\min_{\sigma(\mathbf{z})} \mathcal{L}_{VAE}(\mathbf{x}) = \frac{1}{S}\sum_{s=0}^{S} \text{MSE}(\mathbf{x}_s, \boldsymbol{\mu}(\mathbf{z}_s)). \tag{3-19}$$

This expression can be effectively integrated into the optimization procedure of the VAE-loss function, even enabling batch-wise estimation of the optimal variance value during the Stochastic Gradient Descent (SGD) process. Further, it should be noted that the expression can also be extended to the multidimensional decoder covariance matrix in a straightforward manner, enabling the use of more complex decoder distributions $p_\theta(\mathbf{x}|\mathbf{z})$ for approximating its true likelihood.

3-3-2 β-VAE

A different approach, called β-VAE, refers to a VAE architecture designed to allow manual tuning of the trade-off between MSE loss and Kullback-Leibler divergence [19]. To achieve this, the hyperparameter β is introduced to the ELBO objective, resulting in the β-VAE loss function $\mathcal{L}^{\beta}_{VAE}$ given by

$$\mathcal{L}^{\beta}_{VAE}(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] - \beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})). \tag{3-20}$$

The distinction in the β-VAE loss function compared to the "traditional" VAE loss function, as introduced in (3-8), is the inclusion of the hyperparameter β.

When comparing β-VAE to the case where the decoder covariance matrix has a shared value for the diagonal elements, represented by the loss function in Equation (3-13), selecting a constant variance of $(\sigma(\mathbf{z}))^2 = \beta$ in the latter case results in a loss function with a weighting ratio between the MSE and Kullback-Leibler divergence terms that match the gradient of the corresponding β-VAE loss function. Therefore, by utilizing an appropriate learning rate, optimizing the β-VAE objective results in the same parameter updates as optimizing the VAE objective with a shared decoder variance value of $(\sigma(\mathbf{z}))^2 = \beta$. In this context, optimizing β-VAE can be seen as manually tuning the shared decoder variance value [34].

By choosing $\beta > 1$, the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ distribution is incentivized to align more closely with the prior $p_\theta(\mathbf{z})$ distribution [19]. In the case of an isotropic centered Gaussian prior $p_\theta(\mathbf{z})$ this implies that the encoder $q_\phi(\mathbf{z}|\mathbf{x})$ is stimulated to learn a more disentangled latent representation of the input data sample $\mathbf{x}$. In this context, a disentangled latent representation refers to a set of minimally correlated latent variables capturing the underlying factors of variation in the data individually. An example of a disentangled feature in an image dataset of faces could be the skin color or the orientation of the face. Generally, disentangled representations are considered interpretable, which in many applications, is a desirable characteristic.

However, β-VAE has the disadvantage of requiring extensive and individual manual tuning of β for each dataset. Even though a disentangled latent representation might be desirable, a model with a high β value may become too focused on minimizing the Kullback-Leibler divergence between the approximate posterior distribution and the prior distribution. This may result in the latent representation being overly constrained and not capturing all of the important information in the data, resulting in poor reconstruction quality. In the extreme case, posterior collapse may occur, which is a phenomenon that will be discussed next.

3-4 Posterior Collapse in VAEs

Posterior collapse is a common issue in VAEs that occurs when the encoder distribution for a subset or all of the latent variables becomes similar or equal to the prior distribution $q_\phi(z_i|\mathbf{x}) \approx p(z_i)$. Consequently, this situation leads to a near zero Kullback-Leibler divergence between these two distributions [28, 29].

When posterior collapse occurs, the learned encoder distribution becomes uniform across all inputs, causing the model to ignore those variables and not store meaningful information in them. As a result, the collapsed variables provide little value for reconstructing the input data or learning useful representations.

To elucidate this concept, consider two different VAEs with a latent space dimension of 2. Both are trained on a subset of the Sussex-Huawei Locomotion (SHL) dataset, for which an initial overview was provided in Section 1-2. However, one of them experiences posterior collapse, while the other does not. This contrasting phenomenon is depicted in Figure 3-3.

One of the causes of posterior collapse can be attributed to the KL-divergence term, which is minimized by the collapsed posterior distribution. Consequently, a high penalty on this term, such as using a large β value in β-VAE, can lead to a global optimum of the ELBO that promotes posterior collapse.
Powerful decoders, on the other hand, can also contribute to posterior collapse. They have high capacity and can closely match the training data, even when the encoder distribution is collapsed. This can lead to the decoder memorizing the training data rather than utilizing meaningful information encoded in the latent variables, ultimately resulting in posterior collapse.
Furthermore, the VAE training process involves optimizing a highly nonlinear objective function with many local minima, which can cause the optimization algorithm to become trapped in one of them. In some instances, these local minima cause collapsed encoder distributions, leading to posterior collapse [28, 29].

Due to the multiple causes of posterior collapse, finding a single solution that effectively prevents it can be challenging. Nonetheless, when working with VAEs, it is crucial to be aware of this phenomenon. Further, it is essential to recognize that a low ELBO value does not necessarily indicate a well-performing VAE, as some or all of its latent variables may still suffer from posterior collapse.

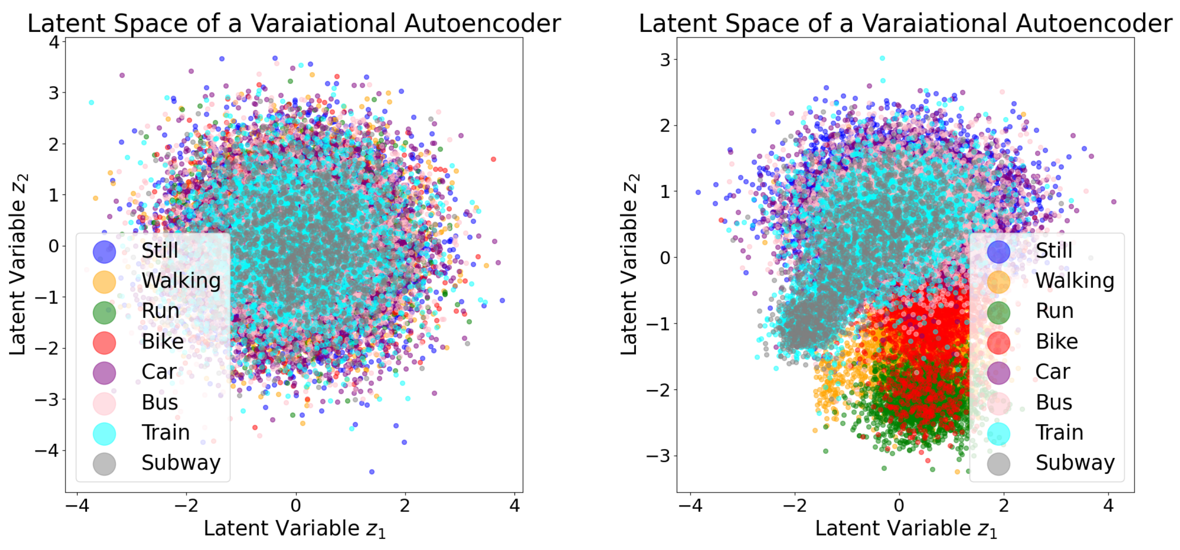

Figure 3-3: Comparison of the 2D Latent Spaces of two distinct VAEs trained on the same subset of the SHL dataset. The VAE generating the left latent space experiences posterior collapse in its latent variables, while the VAE generating the right does not. Further, the distinct colors signify the class of the corresponding latent representation.

Chapter 4

Variational Autoencoder for Data Augmentation

This chapter aims to provide an understanding of the Variational Autoencoder (VAE) based data augmentation technique called Iterative Hierarchical Data Augmentation (IHDA), which is studied throughout this thesis. The chapter begins by discussing common data augmentation techniques for time series data and their limitations, which serve as motivation to explore alternative approaches, such as generative models like VAEs. The chapter then describes how VAEs can be utilized for data augmentation, leading to the introduction of the IHDA algorithm.

4-1 Data Augmentation for Time-Series Data

Data Augmentation (DA) is a technique used to artificially increase the size of a dataset by creating new, modified samples of existing data samples by applying various transformations to them. This can be useful for Machine Learning (ML) and Deep Learning (DL) models, as it increases the amount and diversity of the training data which may improve the model's ability to generalize and perform well on data not utilized during training [21, 47].

Several methods exist to augment time-series data, of which two prevalent techniques are "Jittering" and "Scaling":

- **Jittering:** This method involves the addition of Gaussian noise to each time step of the time series data. Jittering can be especially advantageous when data variations mainly stem from differing noise levels. For instance, in the context of transportation mode detection (TMD), sensors embedded in different devices may introduce varying degrees of noise [21, 47]. Consequently, employing jittering can effectively yield a robust dataset that captures the distinct noise levels associated with sensors in various devices.

- **Scaling:** This technique adjusts the magnitude of time series data samples by multiplying each element by a scaling parameter. By changing the amplitude of the time series data, this technique can simulate variations in intensity conditions [21, 47]. For example, in TMD, the same pattern across the time axis might exist for a specific transportation mode, but at different intensity levels. Through scaling, these varying intensity levels can be artificially replicated, thereby creating a dataset that represents the wide range of intensity levels encountered in real-world transportation modes.

While both techniques are utilized and have been proven to enrich data diversities, they come with certain limitations. Most notably, these methods lack the ability to capture or generate complex dependencies and correlations between different time steps or data dimensions, such as interdependencies between various sensor streams.

Generative models, including the VAEs offer an appealing alternative in this context. These models are designed with the intention to learn the underlying data distribution of the dataset. Subsequently, their design aims to enable them to generate new, realistic data instances that go beyond the simplicity of merely modifying existing samples. As such, the potential of these models for advanced data augmentation offers exciting prospects. Consequently, the following sections will delve into the application of VAE for data augmentation in a more detailed manner.

4-2 Data Augmentation with VAEs

Variational Autoencoder (VAE) introduce two essential methodologies for DA [35]. Each approach uses the principle of transforming latent space representations into new data samples, but the source of these representations varies.

The first method involves sampling a latent space representation $\mathbf{z}$ from the prior distribution $p_\theta(\mathbf{z})$, which is then fed into the decoder network $p_\theta(\mathbf{x}|\mathbf{z})$ to generate a new data sample $\hat{\mathbf{x}}$. However, the quality of the generated sample $\hat{\mathbf{x}}$ depends on the accuracy of the prior $p_\theta(\mathbf{z})$ in approximating its true distribution $p_{\theta^*}(\mathbf{z})$. If their difference is significant, sampling from $p_\theta(\mathbf{z})$ may result in unrealistic generated data samples $\hat{\mathbf{x}}$.

In contrast, the second method involves sampling a latent representation $\mathbf{z}$ from the learned posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$ given an input sample $\mathbf{x}$, and then mapping it back to the observation space by utilizing the decoder network. Unlike the first method, this technique enables the generation of a new data sample similar to the specific inputted data sample $\mathbf{x}$. Another advantage is that feeding one data sample $\mathbf{x}$ multiple times to the same VAE architecture will produce a set of reconstructed samples that are all distinct from each other but still resemble the input signal $\mathbf{x}$. In light of the latter VAE-based DA method, the next section introduces the IHDA algorithm.

4-3 Post-training Iterative Hierarchical Data Augmentation for Deep Networks

The Post-training Iterative Hierarchical Data Augmentation (IHDA) algorithm is a Data Augmentation technique for Deep Learning models, developed by Khan and Fraz [22] and presented at the Conference on Neural Information Processing Systems (NeurIPS). The IHDA algorithm leverages the two properties of VAEs: the ability to represent data samples as distributions, facilitating the generation of similar but distinctive versions of a given sample, and the ability to reduce the dimensionality of data in a meaningful way.

As the name suggests, the IHDA algorithm is applied post-training of the representation learning function $F_{\psi}^{L} : \mathbf{x}_k \mapsto \hat{y}_k$, which transforms the input $\mathbf{x}_k$ to its corresponding feature representation $\hat{y}_k$. The dataset used for training F_{ψ}^{L} is denoted as $\mathbf{X} = \{\mathbf{x}_k, y_k\}_{k=1}^{K}$, while $\mathrm{X} = \{\mathbf{x}_k\}_{k=1}^{K}$ denotes the dataset containing only the input training samples for differentiation. Further, in the context of this thesis, F_{ψ}^{L} represents a transportation mode classifier that maps mobile phone sensor data-based features to the associated transportation mode label. This network, defined by its parameters ψ, is composed of L layers. Moreover, $\mathbf{x}^{l}$ signifies the input of the l-th layer, obtained by mapping the input $\mathbf{x}$ through the first $l-1$ layers of the representation learning function F_{ψ}^{L}, while $\mathbf{X}^{l} = \{\mathbf{x}_k^{l}, y_k\}_{k=1}^{K}$ and $\mathrm{X}^{l} = \{\mathbf{x}_k^{l}\}_{k=1}^{K}$ represent the corresponding dataset.

The IHDA algorithm is an iterative algorithm, meaning that the augmentations at level $l \in L$ are used to fine-tune all subsequent layers before performing the augmentation at level $l+1$. This process is repeated for each layer until all layers have been augmented. Fine-tuning, in this context, is interpreted as the utilization of an augmented dataset to make adjustments to the targeted layers. This involves updating the parameters of the considered layer l and its subsequent layers, which define their transformations and were initially set during the training phase. Next, the data augmentation technique used to augment the data at layer l is explained.

Data Augmentation at Layer l

The initial step in applying the IHDA algorithm to layer l is the generation of dataset X^{l}, accomplished by mapping all samples from X through the first $l-1$ layers of representation learning function F_{ψ}^{L}.

Once the dataset X^{l} has been established, it becomes the training set for a VAE composed of an encoder $q_{\phi}(\mathbf{z}|\mathbf{x}^{l})$ and a decoder $p_{\theta}(\mathbf{x}^{l}|\mathbf{z})$, which are trained by optimizing the VAE loss function $\mathcal{L}_{VAE}(\mathrm{X}^{l})$. Khan and Fraz [22] specified utilizing the simplified loss function, given by (3-17), which combines Mean Square Error (MSE) and Kullback-Leibler divergence, though it omits the determination of an appropriate weighting between these terms, as discussed in Section 3-3. Following training completion, the VAE is utilized for subsequent steps.

The IHDA algorithm's objective is to sample new data samples only for regions where classification is considered to be challenging. To achieve this, the algorithm exploits the dimensionality reduction capability of the VAE and employs its latent space to identify such regions. Accordingly, the IHDA algorithm utilizes the encoder $q_{\phi}(\mathbf{z}|\mathbf{x}^{l})$ to map the entire dataset X^{l} to its latent space representations following (3-15), yielding the dataset $\mathrm{Z}^{l} = \{\mathbf{z}_k^{l}\}_{k=1}^{K}$ that contains one latent vector $\mathbf{z}^{l}$ for each sample $\mathbf{x}^{l} \in \mathrm{X}^{l}$.

To determine whether the representation learning function F_{ψ}^{L} might benefit from sampling additional data of a considered latent representation $\mathbf{p} \in \mathrm{Z}^{l}$, the IHDA algorithm computes a potential for each sample $\mathbf{p} \in \mathrm{Z}^{l}$. As a preliminary step, the IHDA algorithm establishes a neighborhood for the considered data sample $\mathbf{p} \in \mathrm{Z}^{l}$. The definition of this neighborhood is meant to encompass samples that are similar to the considered sample $\mathbf{p}$, as defined by the following equation

$$\mathcal{N}_{\mathbf{p}} = \{\mathbf{q} \in \mathrm{Z}^l \mid \mathrm{dist}(\mathbf{p}, \mathbf{q}) \leq w\}. \tag{4-1}$$

Here, $\mathrm{dist}(\mathbf{p}, \mathbf{q})$ is the distance function between the latent vectors $\mathbf{p}$ and $\mathbf{q}$, and the hyperparameter w specifies the maximum distance within which samples $\mathbf{q}$ are considered neighbors of sample $\mathbf{p}$. Khan and Fraz [22] proposed to utilize as the distance metric the cosine similarity matrix $S_c(\mathbf{p}, \mathbf{q})$ which is defined as

$$S_c(\mathbf{p}, \mathbf{q}) = \frac{\mathbf{p} \cdot \mathbf{q}}{||\mathbf{p}||||\mathbf{q}||}, \tag{4-2}$$

where $\mathbf{p} \cdot \mathbf{q}$ is the dot product of the two vectors and $|| \cdot ||$ is the L2 norm. The cosine similarity matrix measures the similarity between two vectors and returns a value between -1 and 1. Two proportional vectors have a cosine similarity of 1, two orthogonal vectors have a similarity of 0, and two opposite directing vectors have a similarity of -1. Importantly, when employing cosine similarity as a distance metric to define the neighborhood of a sample $\mathbf{p}$, defined by (4-1), the condition changes to $\mathrm{dist}(\mathbf{p}, \mathbf{q}) \geq w$, since a larger cosine similarity value indicates greater similarity between two compared samples, which is contrary to other similarity metrics such as the Euclidean distance.

The potential measure is introduced to determine if a considered latent vector $\mathbf{p}$ in its neighborhood $\mathcal{N}_{\mathbf{p}}$ is surrounded by more instances of other classes than its own. This measure is calculated using the radial basis function (RBF) given by

$$\mathrm{RBF}(\mathbf{p}, \mathbf{q}) = e^{-(\frac{||\mathbf{p}-\mathbf{q}||}{\gamma})^2}. \tag{4-3}$$

The potential measure is determined by adding the value of the RBF for a sample $\mathbf{p}$ and its neighbor $\mathbf{q}$ to the potential if they belong to different classes, and subtracting it otherwise. This is illustrated in Algorithm 1. A positive potential value indicates that $\mathbf{p}$ is surrounded by more instances of other classes than its own, and hence $\mathbf{p}$ is selected for generating new data.

The hyperparameter γ controls the influence of close neighbors on the potential calculation compared to distant ones. Empirically, a value of $\gamma = 0.05$ has been shown to perform well, which was reported by Khan and Fraz [22]. Only samples with positive potential and for which at least one neighboring sample belongs to the same class are considered potential samples in order to eliminate potentially noisy data.

Once the potential samples are identified in the latent space, their corresponding input representation is used to create the final set of latent space distributions for generating artificial samples for DA at step l of F_{ϕ}^{L}. The set of distributions is represented as

$$\mathcal{P}^l = \{(\boldsymbol{\mu}(\mathbf{x}^l), \boldsymbol{\Sigma}(\mathbf{x}^l)) = q_{\phi}(\mathbf{z}|\mathbf{x}^l) \,|\, \mathbf{x}^l \in \mathrm{X}^l, \, \mathrm{potential}(\mathbf{x}^l > 0)\}. \tag{4-4}$$

This distribution set is then used to produce an augmented dataset at step l in the following manner

Algorithm 1 The algorithm to compute potential of a point $\mathbf{p} \in \mathrm{X}^l$

```
1: Input: Point $\mathbf{p}$, neighborhood $\mathcal{N}_p$, and the spread of RBF $\gamma$
2: potential = 0
3: noisy = true
4: for every $\mathbf{q} \in \mathcal{N}_p$ do
5:     if $\mathbf{p}$ and $\mathbf{q}$ have the same class label then
6:         noisy = false
7:         potential = potential - $e^{(\frac{\|\mathbf{p}-\mathbf{q}\|}{\gamma})^2}$
8:     else
9:         potential = potential + $e^{(\frac{\|\mathbf{p}-\mathbf{q}\|}{\gamma})^2}$
10: if noisy then
11:     potential = -1
12: Output: potential
```

$$\hat{\mathbf{X}}^l = \{(\hat{\mathbf{x}}^l, y) | \hat{\mathbf{x}}^l = q_\theta(\mathbf{x}|\mathbf{z}),\ \mathbf{z} = \boldsymbol{\mu}(\mathbf{x}) + \boldsymbol{\Sigma}^{\frac{1}{2}}(\mathbf{x}) \odot \boldsymbol{\epsilon},\ (\boldsymbol{\mu}(\mathbf{x}^l), \boldsymbol{\Sigma}(\mathbf{x}^l)) \in \mathcal{P}^l,\ \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)\}, \quad (4\text{-}5)$$

where y is the class label associated with $\mathbf{x}$ whose corresponding distribution is $(\boldsymbol{\mu}(\mathbf{x}^l), \boldsymbol{\Sigma}(\mathbf{x}^l)) \in \mathcal{P}^l$. For each $(\boldsymbol{\mu}(\mathbf{x}^l), \boldsymbol{\Sigma}(\mathbf{x}^l)) \in \mathcal{P}^l$, a random number of 3, 5, or 10 new data samples are generated. After generating $\hat{\mathbf{X}}^l$, the algorithm fine-tunes layer l and all subsequent layers by training the model on the augmented dataset for a few epochs. Then, the entire process is repeated for the next layer, $l+1$, resulting in the IHDA algorithm. The algorithm is denoted as:

Algorithm 2 The IHDA Algorithm

```
1: Input: Dataset $\mathbf{X}$, and distance $w$
2: Train $F_\psi^L$ on $\mathbf{X}$
3: for $l = 1$ to $L$ do
4:     Map $\mathbf{X}$ to $\mathbf{X}^l$ using $F_\psi^L$
5:     Train VAE on $\mathbf{X}^l$ using Equation (3-17)
6:     Compute $\mathcal{P}^l$ using Equation (4-4)
7:     Generate $\hat{\mathbf{X}}^l$ using (4-5)
8:     Use to $\hat{\mathbf{X}}^l$ fine-tune layers $\{l, l+1, \ldots, L\}$ of $F_\psi^L$
9: Output: $F_\psi^L$
```

Chapter 5

Methodology

This chapter aims to provide a comprehensive understanding of the key purposes of this thesis, which where outlined in Section 1-4.

The primary objective is the practical implementation of the Iterative Hierarchical Data Augmentation (IHDA) as proposed originally by Khan and Fraz [22]. This necessitates an exploration of various fundamental elements such as the dataset, inferred features, and the structure of the neural networks utilized. In addition to this, the chapter supplies detailed insights into the actual implementation process, such as the selected hyperparameters and the hardware deployed.

The secondary objective focuses on the identification and thorough examination of potential enhancements to the IHDA. These proposed improvements will be described and justified providing readers a complete understanding of the motivations behind and potential benefits of these modifications.

5-1 Dataset, Preprocessing, Classifier Architecture, and Evaluation

Delving into the first objective, this section deals with the practical implementation of the IHDA algorithm as proposed by Khan and Fraz [22]. In their work, the authors evaluated the performance of the IHDA algorithm in enhancing the classifier's performance using the classifier outlined by Wang et al. [42]. For consistency, the same classifier is utilized in this thesis.

Following the Machine Learning (ML) workflow outlined in Section 2-1, this section describes the steps of data acquisition and preprocessing, feature extraction and engineering, and model selection. Each step of this workflow is integral to the successful replication of the results obtained by Khan and Fraz [22], and therefore will be discussed in this section.

The classifier employs the Sussex-Huawei Locomotion (SHL) dataset, acquired and preprocessed by Gjoreski et al. [13]. The feature extraction, engineering, and model selection were performed by the classifier's authors [42], while the model selection for the Variational Autoencoder (VAE) segment of the IHDA was completed by Khan and Fraz [22].

As no modifications have been made to the steps proposed by Gjoreski et al. [13], Wang et al. [42], and Khan and Fraz [22], detailed insights of each process can be found in their respective references: [13] for the SHL dataset, [42] for the classifier, and [22] for the VAE architecture integrated into the IHDA algorithm.

5-1-1 Data Acquisition and Preprocessing: The SHL Dataset

This study utilizes a subset of the SHL dataset. The complete SHL dataset consists of 2812 hours of labeled transportation mode detection (TMD) data, collected in 2017 from three participants, each carrying four smartphones at distinct body positions: Bag, Hips, Torso, and Hand [13]. Each individual was equipped with four smartphones located at four distinct body positions: Bag, Hips, Torso, and Hand. The dataset covers eight transportation modes: Still, Walk, Run, Bike, Bus, Car, Train, and Subway [13]. A brief overview of the dataset was provided in Section 1-2, and a more comprehensive exploration can be found in [13].

The utilized subset was released as part of the 2018 SHL recognition challenge and is considered, as it was utilized by Wang et al.[42] for developing the classifier used in this thesis. The subset includes recordings from the first participant, with the smartphone located at the Hips position. It comprises data from sensors: accelerometer, gyroscope, magnetometer, linear acceleration, gravity, orientation, and ambient pressure. However, only the accelerometer, gyroscope, and magnetometer were utilized in the development of Wang et al.'s [42] classifier, hence, only these were considered within the scope of this thesis. Further, it's important to note that each of these sensors provides three sensor streams corresponding to measurements in the x, y, and z directions of the device.

The subset consists of 366 hours of recordings, which are further divided into a training dataset comprising 271 hours and a testing dataset comprising 95 hours. However, it is important to note that the data is not uniformly distributed among the different transportation modes, as illustrated in Figure 5-1.

Furthermore, the data is not available in its chronological sequence. The SHL challenge organizers processed the training and testing datasets by dividing them into segments using a one-minute non-overlapping sliding window. The order of the segments was randomly permuted, resulting in $16,310$ segments in the training dataset and $5,698$ segments in the testing dataset. Each segment contained $6,000$ data samples for every sensor stream of the accelerometer, gyroscope, and magnetometer, given that the sensors sampled data at a frequency of $f_s = 100\text{Hz}$.

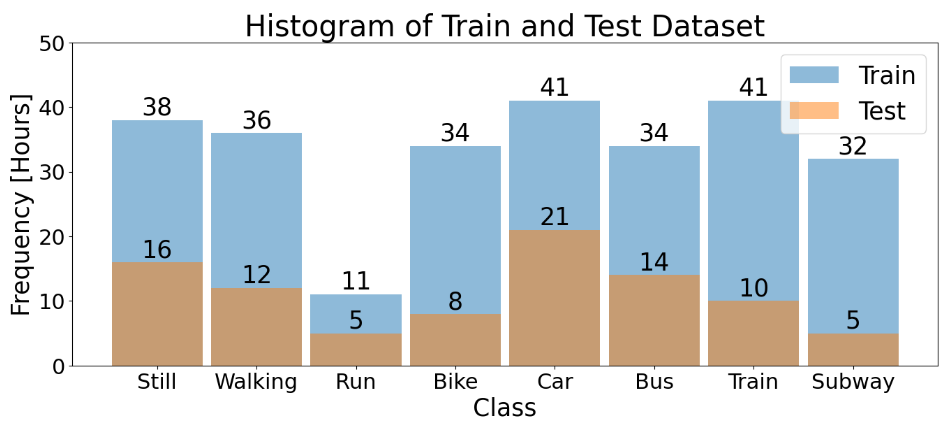

Figure 5-1: Data distribution across various modes of transportation within the utilized SHL dataset subset for training and testing.

5-1-2 Feature Extraction and Engineering

In the preceding section, it was established that the training and testing data samples for each sensor stream are organized in one-minute intervals, with each frame encompassing 6000 samples, a result of the sensors' sampling frequency of $f_s = 100$ Hz. These samples serve as the basis for feature engineering and extraction, which are utilized in training the classifier proposed by Wang et al. [42].

The initial step in the feature extraction process involves dividing the one-minute segments from the SHL dataset into smaller data frames using a sliding window technique.

For the training dataset, a window length of 5 seconds and a skip size of 2.5 seconds are used, resulting in 375,130 frames for each of the considered sensor streams. The testing dataset follows a similar approach, with a window length of 5 seconds and a skip size of 5 seconds, resulting in 68,376 frames. Consequently, each frame for every sensor stream contains samples recorded over a 5-second duration, equivalent to 500 samples. These 5-second frames serve as the basis for subsequent feature extraction and engineering steps.

The accelerometer, gyroscope, and magnetometer sensors were utilized for the recognition task, with each sensor providing three sensor streams representing measurements along the device's x-, y-, and z-axes. To make the features independent of the smartphone's unknown pose and orientation, the magnitudes of the three channels for each sensor were calculated as

$$\begin{aligned} Acc &= \sqrt{Acc_x^2 + Acc_y^2 + Acc_z^2}\,, \\ Gyr &= \sqrt{Gyr_x^2 + Gyr_y^2 + Gyr_z^2}\,, \\ Mag &= \sqrt{Mag_x^2 + Mag_y^2 + Mag_z^2}\,. \end{aligned} \tag{5-1}$$

A Fourier transform was then applied to the frames containing the magnitudes of the inertial sensors given by (5-1). The magnitudes of the Fourier transformation for frequencies in the range $[0, 0.5f_s]$ were retained and used for further data processing. This resulted in a feature vector $\boldsymbol{f}_k$ belonging to the k-th frame and denoted as

$$\boldsymbol{f}_k = \begin{bmatrix} \boldsymbol{f}_k^{acc} & \boldsymbol{f}_k^{gyr} & \boldsymbol{f}_k^{mag} \end{bmatrix}, \tag{5-2}$$

where $\boldsymbol{f}_k^{acc}$, $\boldsymbol{f}_k^{gyr}$ and $\boldsymbol{f}_k^{mag}$ denote the features belonging to the accelerometer, gyroscope, and magnetometer, respectively. Each of these feature vectors is of size $\mathbb{R}^{1\times 251}$, hence resulting in the feature vector $\boldsymbol{f}_k \in \mathbb{R}^{1\times 753}$.

The final processing step involved normalizing the features to a range of $[0, 1]$. Suppose the i-th feature of the k-th frame of the accelerometer is denoted by f_i^{acc}, where the subscript k is omitted for readability. Further, Q_i^{95} and Q_i^{5} represent the 95th and 5th percentiles of f_i^{acc} across all the frames in the training data. The normalization of each frame is then performed as

$$\bar{f}_i^{acc} \leftarrow \min\left(\max\left(\frac{f_i^{acc} - Q_i^5}{Q_i^{95} - Q_i^5}, 0\right), 1\right). \tag{5-3}$$

After the same normalization procedure was performed for the gyroscope and magnetometer features, the testing dataset features were normalized using Q_i^{95} and Q_i^5, which were computed beforehand from the training dataset. The resulting feature vector for frame k in both the training and testing datasets is denoted as $\bar{\boldsymbol{f}}_k$, which will serve as input $\mathbf{x}_k$ for the network architectures introduced later on.

It should be noted that normalization, in this context, refers to the scaling of feature values to a common range of [0,1]. This is performed across the 'feature' axis, meaning each feature is normalized independently based on its own distribution. However, this operation does affect the inter-feature dependencies along the 'sample' axis, particularly across the differing Fourier magnitude values within a given sample, as different features of a sample may have varying scaling factors applied. Consequently, this added complexity further challenges human interpretation of the data.

In sum, the features produced by this extraction and engineering process are rooted in 5-second measurements of the accelerometer, gyroscope, and magnetometer. For each of these three sensors, the magnitude of their three streams in the x, y, and z directions is computed. These time-domain features are subsequently transformed into the frequency domain using Fourier transformation, with the magnitudes of these transformed signals retained and amalgamated into a single sample. The final step of the feature engineering process involves normalizing the features to a range of 0 to 1, yielding the definitive features that will be inputted into the classifier.

5-1-3 Model Selection: Benchmark Classifier

The previous section outlined the process of feature extraction and engineering, resulting in a feature set that will be used to develop the classifier proposed by Wang et al.[42]. In this thesis, this classifier will be referred to as the benchmark classifier, serving as a baseline for evaluating the performance enhancement achieved by the IHDA algorithm.

The benchmark classifier represents a Dense Neural Network (DNN) that receives the engineered features as input and processes them through three hidden layers and one output layer. Each hidden layer consists of a dense layer followed by a batch normalization layer, a Rectified Linear Unit (ReLU) activation layer, and a dropout layer.

The batch normalization layer normalizes each input channel across a mini-batch, with its objective to accelerate the training process [20]. Additionally, the dropout layer randomly sets input elements to zero with a probability of 25% aiming to prevent overfitting [37].

The output layer comprises a dense layer, a SoftMax layer, and a classification layer responsible for inferring the transportation mode of the current frame. For detailed explanations of the activation functions used, specifically ReLU and SoftMax, please refer to Section 2-4-2.

Additional specifications of the classifier are summarized in Table 5-1, and Figure 5-2 provides a schematic representation.

Input $\mathbf{x}_k$	$\bar{\boldsymbol{f}}_k$
Hidden Layer	Number of Layers = 3
	Number of Nodes per Layer = 500
	Dropout Ratio = 25%
Batch Size	500

Table 5-1: Architectural specifications of the benchmark classifier.

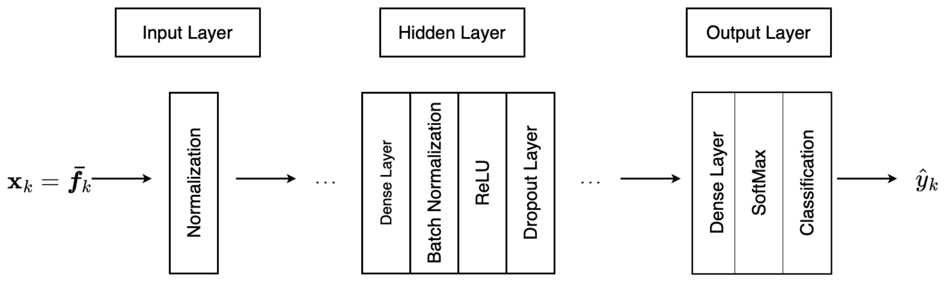

Figure 5-2: Schematic representation of the benchmark dense neural network classifier, wherein the input $\mathbf{x}_k$ corresponds to the normalized features $\bar{\boldsymbol{f}}_k$ derived from the feature extraction and engineering phase, and $\hat{y}_k$ represents the predicted output label.

Loss Function

The Softmax activation function, previously introduced and integral to the output layer of the benchmark classifier, warrants further discussion in the context of the loss function. Given the SHL dataset, which includes data from eight transportation modes, the output of the Softmax function of the benchmark classifiers becomes an eight-dimensional vector. Each dimension p_{kc} then serves to estimate the probability that the k-th input sample belongs to class c, facilitating the interpretation of each output as the likelihood of an input belonging to a particular class, as detailed in Section 2-4-2.

When training an architecture that utilizes the Softmax layer as the output layer, the categorical cross-entropy loss function is a common choice. This approach was also employed by Wang et al.[42] for the benchmark classifier.

The categorical cross-entropy loss function assesses the model's performance based on the discrepancy between the predicted probabilities and the true labels. For a dataset with K instances and C unique classes, the categorical cross-entropy loss is expressed as

$$L = -\frac{1}{K}\sum_{k=1}^{K}\sum_{c=1}^{C} y_{kc}\log(p_{kc}), \tag{5-4}$$

where y_{kc} represents the ground truth, which is equal to one when the instance k belongs to class c and is zero in all other cases. On the other hand, p_{kc} represents the model's predicted probability that instance k belongs to class c, obtained from the output of the Softmax function. The objective during training is to minimize this loss, thereby aligning the model's predicted probabilities as closely as possible with the true labels.

Assessing Classifier Performance with Accuracy and F1 Score

Following the training phase wherein the loss function is minimized, Wang et al. [42] utilizes both accuracy and F1 score to evaluate the benchmark classifier's performance. Accuracy quantifies the proportion of correctly classified instances relative to the total number of instances [6, 16]. It is defined as the ratio of correctly classified instances to the total instances and can be computed as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP+TN}{TP+TN+FP+FN}. \tag{5-5}$$

In this equation, TP, FP, TN, and FN represent the number of true positives, false positives, true negatives, and false negatives, respectively. Accuracy ranges from 0 to 1, with higher values signifying better performance.

However, accuracy alone may not always provide a comprehensive assessment of a classifier's performance, particularly when dealing with imbalanced datasets. In such cases, a classifier predicting the majority class for all samples could still achieve high accuracy if the majority class dominates the dataset.

This is where the F1 score comes into play. Like accuracy, the F1 score also ranges from 0 to 1, with higher values signifying better performance. The F1 score offers a more reliable performance metric for imbalanced datasets by combining precision and recall. Precision is defined as the ratio of true positive classifications to the total number of positive classifications, while recall is the ratio of true positive classifications to the total number of actual positives. The F1 score is computed as the harmonic mean of precision and recall and can be determined by

$$\text{Precision} = \frac{TP}{TP+FP}, \qquad \text{Recall} = \frac{TP}{TP+FN}, \qquad \text{F1} = \frac{2\times\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}. \tag{5-6}$$

5-1-4 Model Selection: IHDA-VAE

Expanding upon the explanation of the benchmark classifier's architecture, the discussion now transitions to the architecture of the VAE utilized by Khan and Fraz [22] in IHDA algorithm.

The encoder consists of two consecutive convolutional layers with dimensions $(64 \times 4 \times 4)$ and $(128 \times 2 \times 2)$, where the values represent the Width, Height, and the number of output channels of the kernel used in the convolutional layer, respectively. A comprehensive explanation of these terms is presented in Section 2-4-1. The output of the second convolutional layer is then passed through two dense layers, which generate the mean and log-variance used to parameterize the encoder distribution.

The decoder follows a mirrored architecture of the encoder. It takes a sampled latent vector $\mathbf{z}$ of dimension 64 as input, which is then passed through a dense layer, followed by two deconvolutional layers with dimensions $(128 \times 2 \times 2)$ and $(64 \times 4 \times 4)$.

Each convolutional and deconvolutional layer is followed by a ReLU activation function and a batch normalization layer. In addition, same padding is employed in all these layers to retain the spatial dimensions of the output. The stride for all convolutional and deconvolutional layers was assumed to be one since it was not specified by Khan and Fraz [22]. A summary of the VAE architecture specifications can be found in Table 5-2.

Component	**Layer**	**Specification**
Encoder Network	Convolutional Layer 1	$128 \times 2 \times 2$ with same padding
	Convolutional Layer 2	$128 \times 2 \times 4$ with same padding
	Dense Layer 1	64 Nodes
	Dense Layer 2	64 Nodes
Decoder Network	Dense Layer	64 Nodes
	Deconvolutional Layer 1	$128 \times 2 \times 4$ with same padding
	Deconvolutional Layer 2	$128 \times 2 \times 2$ with same padding
Activation Function	ReLU	
Batch Normalization	After each Convolutional and Deconvolutional Layer	

Table 5-2: Detailed architectural specifications of the IHDA-VAE encoder and decoder networks.

5-2 Implementation Details

This section offers a closer look at the practical implementation of the discussed methods, namely the benchmark classifier and the IHDA algorithm. It provides supplemental information about the derivation of hyperparameters, along with details about the hardware and optimization algorithm used during the implementation.

5-2-1 Benchmark Classifier Implementation

The benchmark classifier was implemented, utilizing the architecture and loss function introduced in Section 5-1-3. The train dataset was divided into a training dataset and a validation dataset using an 80/20 split ratio. Stratified sampling was applied for splitting the data, ensuring that the class distribution in both the training and validation datasets maintains consistency with the original dataset. The classifier was then trained for 700 epochs, utilizing a learning rate of $\eta = 10^{-4}$.

5-2-2 IHDA Algorithm Implementation

The IHDA algorithm's implementation used the same stratified training and validation datasets, as prepared for the benchmark classifier. The specific steps involved in the IHDA algorithm are described in detail in Section 4-3, but a brief overview of the main steps of applying the IHDA to layer l of the considered classifier will be provided here:

1. **Training the VAE architecture:** The VAE architecture, as specified in Section 5-1-4, is trained on the input of the corresponding hidden layer l of the classifier. The training process involved training the VAE for 30 epochs using a learning rate of $\eta = 3 \cdot 10^{-5}$, following the approach outlined by Khan and Fraz [22].

2. **Hyperparameter search for $\boldsymbol{w}$:** A hyperparameter search is conducted to determine the optimal distance hyperparameter w, which defines the distance threshold within which other latent vector samples are deemed to be in the neighborhood of a given latent space vector. For a detailed description of this neighborhood $\mathcal{N}_{\mathbf{p}}$, refer to (4-1) and Section 4-3. Grid search is applied with w values ranging from 0 to 1, with a step size of 0.05, which is in line with the approach followed by Khan and Fraz [22]. For each w value, positive potential samples from the training dataset are identified, and their reconstructions are generated using the trained VAE. The benchmark classifier is then retrained for 30 epochs, utilizing these reconstructed samples. During the retraining process, the learning rate is set to $\eta = 10^{-5}$ which is 10 times smaller than the learning rate used during the training of the benchmark classifier. This adjustment is motivated by the concept of "fine-tuning" described by Khan and Fraz [22], which in this thesis is interpreted as applying minor adjustments to the weights and biases of the trained benchmark classifier to optimize its performance. It is important to note that retraining involves adjusting the weights and biases of the considered hidden layer and its consecutive layers, as recommended by Khan and Fraz [22]. The goal is to find the w value that yields the lowest validation loss of the benchmark classifier during this retraining process, utilizing the same validation dataset used for training the benchmark classifier. The w value that achieves the lowest validation loss during retraining is selected as the optimal hyperparameter.

3. **Updating the benchmark classifier weights and biases:** If the retrained classifier with the lowest validation loss was lower compared to that of the benchmark classifier, the adjusted weights and biases are used to update the benchmark classifier, which is then used during the hyperparameter search for w of the consecutive hidden layer.

4. **Proceeding to the next hidden layer:** The complete process is repeated for the subsequent hidden layer $l + 1$, following the same steps of training the VAE, conducting the hyperparameter search, and updating the benchmark classifier weights and biases.

5-2-3 Hardware and Optimization Algorithm

The implementation of all algorithms discussed in this thesis utilized PyTorch 2.0.1, an open-source machine learning library developed by Meta AI and currently maintained by the Linux Foundation [32]. Google Colab served as the platform for this implementation, facilitating the use of the NVIDIA T4 Tensor Core GPU in combination with CUDA 11.8.89 to perform computations.

For network parameter optimization of the proposed models, the Adam algorithm was employed [23]. Adam represents a variant of the Stochastic Gradient Descent (SGD) algorithm discussed in Section 2-3-2. The key distinction of Adam is its adaptive learning rate adjustment for each model parameter. This adaptability is achieved by estimating the first and second moments of the gradients. By integrating these moment estimates, Adam dynamically regulates the learning rate throughout the optimization process, which can enhance convergence efficiency [23].

5-2-4 Computational Limitations

The application of the IHDA algorithm, particularly the methodology used to calculate a sample's potential, brings forth several computational challenges. As detailed in Algorithm 1(Section 4-3), it involves identifying the neighbors of each sample in the dataset, which necessitates computing the distance between the sample in question and every other sample within the dataset.

This operation has an inherent computational complexity of $O(n^2)$, which rapidly scales with the size of the dataset. Given the dimensions of the dataset, the cost of this approach becomes readily apparent. Despite the utilization of high-performance NVIDIA T4 Tensor Core GPU for computations, during experimental runs, it was observed that these distance calculations were exceedingly time-consuming. In some instances, they even led to an overconsumption of available system memory (RAM), causing system failure. This issue may be compounded by inefficient memory management, a factor that was not optimized during the course of this thesis.

In the results section (Section 6), numerous decisions were necessitated by these computational constraints. These decisions underscore that the practical implementation of the algorithm was constrained by the intensive computational requirements and memory demands associated with determining sample potentials.

5-3 Proposed Enhancements to the IHDA Algorithm

As the central objective of this thesis, this section presents two potential enhancements to the IHDA algorithm. These enhancements, derived from a theoretical analysis of the IHDA algorithm, could potentially improve its performance and applicability:

> **Proposal 1:** Optimizing the Relative Contributions of the Kullback-Leibler (KL) Divergence and Mean Square Error (MSE) Terms in the VAE Loss Function.
>
> **Proposal 2:** Integrating the Kullback-Leibler (KL) Divergence as a Reliable Latent Space Similarity Metric in the IHDA Algorithm.

This section elaborates on these proposed enhancements and provides rationale as to why the IHDA algorithm could potentially benefit from these improvements.

5-3-1 Proposal 1: Rebalancing the VAE Loss Function

Proposal 1:

> Optimizing the Relative Contributions of the Kullback-Leibler (KL) Divergence and Mean Square Error (MSE) Terms in the VAE Loss Function.

Justification: The first proposed enhancement arises from examining the VAE loss function, as implemented by Khan and Fraz [22], presented in Equation (3-17). This function is essentially the sum of two components: the Kullback-Leibler (KL) divergence term and the Mean Square Error (MSE) term. The KL divergence term penalizes deviations in the encoder distribution $q_\phi(\mathbf{z}|\mathbf{x})$ from the assumed latent space prior distribution $p_\theta(\mathbf{z})$, while the MSE term penalizes discrepancies between the input data and its reconstruction. Detailed explanations can be found in Section 3-2.

In the context of this thesis, this loss function was introduced in Section 3-3 as the “simplified” VAE loss function due to its implicit assumption of equal weight of one for both the KL divergence and the MSE terms, which may lead to suboptimal VAE performance.

As discussed in Section 3-3, the balance between the MSE and KL divergence terms in the VAE loss function is critical for the performance of the resulting VAE. For instance, prioritizing the KL divergence term could lead to its minimization at the expense of an elevated MSE term. This is far from ideal as a low KL divergence term could signify a large proportion of posterior collapsed latent variables, and a high MSE term implies poorly reconstructed input samples. On the other hand, a lower weight on the KL divergence term relative to the MSE term might yield a low MSE term, indicating well-reconstructed input data. However, this condition could coexist with a high KL divergence term, reflecting a significant deviation of latent variable distributions from the intended latent space distribution.

Excessive divergence is undesirable, particularly in the context of implementing the VAE within the IHDA algorithm. Broad dispersion of the latent variables across the latent space could reduce the effectiveness of comparing the resulting latent vectors using similarity metrics, such as Cosine Similarity or Manhattan distance, as proposed by Khan and Fraz [22]. Extreme values from outlier latent variables could dominate the contributions of other variables. Consequently, the optimal weighting of the MSE and KL divergence terms in the VAE loss function demands thorough reconsideration to ensure a more effective weighting strategy.

Thus, the first enhancement aims to refine the balance between the MSE and KL divergence terms to augment the VAE performance within the IHDA algorithm. Ideally, achieving a balanced weight ratio could result in two improvements:

1. **Enhanced Reconstruction Capacity:** A refined balance could enable the resulting VAE to generate more precise reconstructions of input samples, potentially enhancing the IHDA algorithm's overall performance, as these reconstructed samples are used for retraining the classifier.
2. **Enhanced Latent Space Mapping:** A refined balance in the weight ratio could contribute significantly by guiding the encoder distribution $q_\phi(\mathbf{z}|\mathbf{x})$ to map input samples to a latent space distribution that approximates the assumed prior latent space distribution $p_\theta(\mathbf{z})$ without inducing a high rate of posterior collapse among latent variables. By ensuring proximity between the encoder distribution $q_\phi(\mathbf{z}|\mathbf{x})$ and the assumed Gaussian prior $p_\theta(\mathbf{z})$, a bounded latent space is maintained, preventing extreme values. By preventing posterior collapse for most latent variables, the latent space variables contain information about their input samples, which is crucial for effective comparison of latent vectors, an integral aspect of the IHDA algorithm.

Section 3-3 discusses how, in a strictly Gaussian VAE, the decoder variance determines the relative weighting between the MSE and KL divergence terms in the loss function which is denoted by (3-13). Two methods for adjusting this decoder variance, namely, the β-VAE and σ-VAE, were introduced in Subsections 3-3-1 and 3-3-2 respectively.

To recap, both methods were introduced with the assumption of a shared decoder variance among all output dimensions. Furthermore, in the context of β-VAE, changing the value of β can be interpreted as manually adjusting the decoder variance, whereas the σ-VAE provides a strategy to determine a suitable decoder variance during the VAE loss function optimization process, thus eliminating the need for manual tuning. The absence of a requirement for manual tuning makes the σ-VAE a more favorable approach compared to the β-VAE.

However, σ-VAE, which was introduced two years prior to the writing of this thesis in 2021, represents a relatively more recent concept compared to the β-VAE, which was unveiled in 2017. To the best of the author's knowledge, its application has not been explored within the TMD field, especially in relation to the SHL dataset. Therefore, this thesis undertakes a comparative analysis between the σ-VAE and β-VAE, which not only puts the performance of σ-VAE into perspective but also aids in deciding whether its performance justifies its implementation in the IHDA algorithm.

5-3-2 Proposal 2: Reevaluating the Distance Metrics

Proposal 2:

> Integrating the Kullback-Leibler (KL) Divergence as a Reliable Latent Space Similarity Metric in the IHDA Algorithm.

Justification: The second potential enhancement for the IHDA algorithm pertains to the process of identifying "positive potential" samples, as discussed in Section 4-3. As outlined, this identification relies on the measure of distance in the latent space between two selected samples, which serves as an index of similarity between the corresponding samples in the input space.

Given the variety of available distance metric, the chosen metric can thus significantly influence the decision on whether further sampling is required for the considered samples, subsequently impacting the overall performance of the IHDA algorithm. In previous work by Khan and Fraz [22], the cosine similarity matrix, along with Euclidean and Manhattan distances, were tested as distance metrics. However, these metrics share a common drawback - their susceptibility to the phenomenon of posterior collapse.

As explained in Section 3-4, posterior collapse of a latent variable occurs when all input samples are mapped to the same latent variable distribution $q_\phi(z_i|\mathbf{x})$, specifically the assumed standard normal distribution for the latent space $p_\theta(z_i)$. In such cases, two similar samples in the input space may be mapped to vastly different regions in the latent space when sampled from the standard normal distribution. Consequently, metrics like Euclidean, Manhattan, and Cosine Similarity can yield significant distances between these samples, despite their similarities in the input space. Conversely, the opposite scenario is also possible. Therefore, when latent variables collapse, they fail to provide meaningful information about their input samples, making them unreliable for inferring similarities between two samples in the input space. Within the examined 64-dimensional latent space, a few collapsed latent variables may not significantly affect the Manhattan or Euclidean distances. However, as the number of collapsed latent variables increases, their influence on the final distance metric, such as the Manhattan or Cosine Similarity metrics used by Khan and Fraz [22], becomes more pronounced. Consequently, the reliability of these metrics as indicators of similarity between two input samples diminishes.

While the first proposed enhancement, detailed in Section 5-3-1, concentrates on fine-tuning the balance between the MSE and KL divergence term in the VAE loss function, which could reduce the occurrence of posterior collapsed latent variables and thus improve the relevance of two samples' similarity in the input space based on their distance in the latent space, this matter continues to demand considerable attention. With a 64-dimensional latent space where each dimension could potentially be subject to posterior collapse, achieving complete prevention of the phenomenon presents a formidable challenge. While preventing posterior collapse is a complex task due to its elusive causality, the proposed solution focuses on accepting that certain latent variables may collapse but employing a distance metric that is less affected by posterior collapse than those used by Khan and Fraz [22].

Therefore as a second potential enhancement to the IHDA algorithm, this thesis proposes an alternative approach, namely comparing the distributions from which the latent variables are derived, instead of directly comparing the variables themselves. In this context, the Kullback-Leibler (KL) divergence, introduced by (3-7), is suggested as a suitable metric for comparing these latent space representations. Similar to the by Khan and Fraz [22] tested metrics, the Kullback-Leibler (KL) divergence increases as the dissimilarity between two distributions increases, indicating a greater difference between them. However, unlike the other metrics, the KL divergence is not negatively influenced by the collapsed latent variables, as will be explained next.

In cases of posterior collapse, where all input samples map to the same standard normal distribution, the Kullback-Leibler (KL) divergence of two samples drawn from the collapsed latent variable distribution will be zero. Hence, if some of the latent variables did not collapse while others did, the collapsed variables do not negatively influence the KL divergence value as they converge to zero. Instead, the non-collapsed variables, which provide information about the similarity of two samples, dominate the similarity metric. Consequently, the KL divergence is largely unaffected by posterior collapse, especially when compared to the previously tested distance metrics. This approach is expected to yield a more reliable assessment of the similarities and differences between latent space samples in the input space.

Summary of Motivation and Anticipated Improvement: The underlying motivation for advocating the use of the KL divergence as a distance metric within the IHDA lies in the quest to increase the robustness of the IHDA algorithm against the phenomenon of posterior collapse in latent space representations. By addressing the limitation of the previously utilized distance metrics, which are affected by collapsed latent variables, the aim is to achieve a more reliable measure of similarity between input samples, thus improving the identification of 'positive potential' samples and, accordingly, the performance of the IHDA algorithm. The KL divergence, which compares the underlying distributions rather than the individual variables themselves, offers a solution to the problem at hand since it remains largely unaffected by posterior collapse.

5-4 Assessing VAEs Reconstruction Proficiency Beyond Traditional Metrics

In order to evaluate the reconstruction performance of the VAEs trained under Proposal 1, an unconventional evaluation metric is deployed which merits some explanation and motivation.

Typically, the evaluation of a VAE's reconstruction performance relies on the measurement of MSE. The MSE serves as an effective measure for tracking the relative improvement of a VAE during training and comparing the reconstruction performance of different VAEs. However, it is incapable of providing an absolute measure of a VAE's reconstruction proficiency due to the fact that the value constituting a “good” MSE threshold is dependent on individual datasets.

To compensate for these limitations, an additional evaluation criterion is introduced. This necessity becomes particularly noteworthy within the framework of the IHDA algorithm. In this context, reconstructed samples play a crucial role in retraining the classifier, thereby making it vital for these samples to retain as many distinguishing features from their input counterparts as possible. Here, distinguishing features are referred to those characteristics that allow differentiation among samples from various classes.

When assessing the VAE's reconstruction performance in the IHDA algorithm, the focus extends to its proficiency in effectively transferring these distinguishing features. It is proposed that the effectiveness of a VAE in performing this task can be evaluated by comparing the benchmark classifier's performance on the reconstructed samples against its performance on the original, non-reconstructed samples.

This comparison serves as a qualitative measure of how accurately the VAE has reconstructed the samples. If the classifier's performance remains consistent between the original and reconstructed samples, it suggests that the VAE has managed to retain the distinguishing features necessary for classification in the reconstruction process. A significant drop in classifier performance, on the other hand, might point to a loss of these features during reconstruction. Consequently, this approach can help in obtaining a more nuanced understanding of a VAE's reconstruction proficiency, beyond what the MSE measure alone can provide.

Chapter 6

Results

This chapter elucidates the results and corresponding analysis of the experiments conducted throughout this thesis, organized in alignment with the research objectives. Initially, it provides an overview of the benchmark classifier performance, followed by its application to the original Iterative Hierarchical Data Augmentation (IHDA) algorithm. Following this, the chapter delves into Proposal 1, a comprehensive comparison between β-VAE and σ-VAE, highlighting the advantages of applying σ-VAE in the IHDA algorithm and its improvements over the model proposed by Khan and Fraz [22]. The focus then shifts to the exploration of cosine similarity and Kullback-Leibler (KL) divergence metrics, evaluating their effectiveness in measuring input sample similarities based on latent space representation. This analysis forms the basis for Proposal 2, our second enhancement. Subsequently, the effectiveness of three combinations of VAE and distance metrics in identifying samples that may require retraining is examined, which further reinforces the validity of the proposed enhancements. Finally, the results from applying the enhancements to the IHDA algorithm are presented, demonstrating the impact of these enhancements.

6-1 Benchmark Classifier

The benchmark classifier, introduced in Section 5-1-3, was trained for 700 epochs. Figure 6-1 displays the training and validation curves, illustrating the changes in loss value, accuracy, and F1 score over the training process. The loss gradually decreases as the number of epochs increases, which aligns with an increase in accuracy and F1 score. Furthermore, all metrics appear to converge after about 100 training epochs.

Table 6-1 provides a summary of the performance of the classifier that achieved the lowest validation loss throughout the 700 training epochs. The recorded evaluation metrics, specifically accuracy and F1 score on the test dataset, are featured under the row titled "Benchmark Classifier 700 Epochs".

For a thorough analysis, the table also incorporates results that were previously reported by Wang et al. [42], labeled as "Reported Results". An examination of these results reveals a

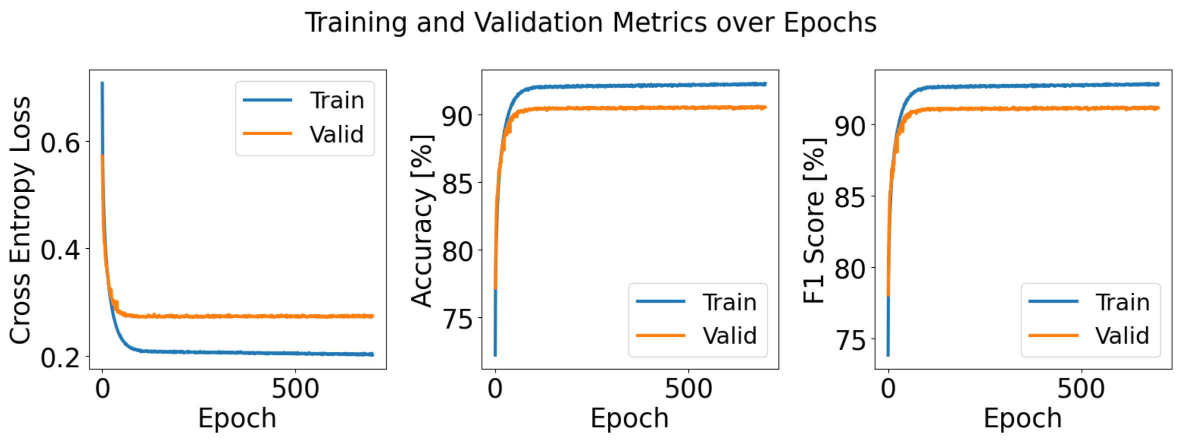

Figure 6-1: Training and validation metrics evolution across 700 epochs for the Benchmark Classifier.

noticeable performance difference in the realms of accuracy and F1 score.

Model	**Test Accuracy** [%]	**Test F1 Score** [%]
Reported Results [42]	82.5	81.7
Benchmark Classifier 700 Epochs	77.62	78.70
5-Fold Cross-Validation	77.35 ± 0.33	78.38 ± 0.22
20%-Classifier	76.13	77.11

Table 6-1: Comparison of the performance of different models on the test dataset using accuracy and F1 score.

When comparing the metrics, "Reported Results" exhibit higher values than those achieved by the "Benchmark Classifier 700 Epochs". The "Reported Results" demonstrate an accuracy of 82.5% and a F1 score of 81.7%, whereas the "Benchmark Classifier 700 Epochs" achieved an accuracy of 77.62% and an F1 score of 78.70%. This deviation, which equates to 4.88% in accuracy and 3% in F1 score, highlights that the "Benchmark Classifier 700 Epochs" did not match the performance of the reported results, necessitating further investigation.

Given the uneven class distribution in the training dataset, as demonstrated in Figure 5-1 and discussed in Section 5-1-1, the observed performance discrepancies could arise from the different data splits used for training and validation. To explore this notion and counter biases emerging from the initial stratified split, a five-fold cross-validation method was employed. This procedure involves the use of random sampling to generate data subsets and enables training and evaluation on five distinct sets of data splits. Each split may exhibit different class distributions in the training and validation datasets, thereby adding variation to the class distribution confronted during the training and validation phases. This method offers insights into the model's sensitivity to fluctuations in class distributions within the training dataset.

Given the time constraints and the observed convergence of training and validation loss at 100 epochs, as illustrated in Figure 6-1, the number of epochs for each fold in the five-fold cross-validation was limited to 200.

In the “5-Fold Cross-Validation” row of Table 6-1, both the average performance and variation in performance across the five models trained during the cross-validation process are presented. These are referred to as mean and standard deviation, respectively. With mean values for accuracy and F1 score being 77.35% and 78.38%, they closely align with those of the “Benchmark Classifier 700 Epochs”. Furthermore, the standard deviations for accuracy and F1 score are relatively small, 0.33% and 0.22%, respectively, when compared to the total metric range from 0% to 100%. Considering the significant difference in performance between the “Benchmark Classifier 700 Epoch” and the “Reported Results”, with a gap of 4.88% in accuracy and 3% in the F1 score, these small standard deviations suggest that the variations in the splits used during training likely do not explain the performance discrepancy observed between the “Benchmark Classifier 700 Epochs” and the results reported by Wang et al. [42]. As of this point in the thesis, the performance divergence is acknowledged, and no further investigations will be pursued.

Table 6-1 additionally features a row labeled “20%-Classifier”, which outlines the performance of a classifier trained using only 20% of the full training dataset. When compared to the “Benchmark Classifier 700 Epochs”, a minor decrease in both accuracy and F1 score for the “20%-Classifier” is observed on the test set, with differences of 1.49% and 1.59% respectively. Given the “20%-Classifier” operated on a considerably smaller segment of the training data, this slight performance reduction is anticipated and considered acceptable.

Given the time and computational constraints discussed in Section 5-2-4, the “20%-Classifier” has been selected for additional investigation in this thesis. The subsequent section will focus on the application of this classifier to the IHDA algorithm, with the aim to assess potential enhancements to the model’s performance.

6-2 Evaluation of IHDA Performance in Classifier Retraining

Table 6-2 showcases the results obtained from retraining the “20%-Classifier” using the IHDA algorithm. It provides the validation loss, accuracy, and F1 score achieved by the retrained classifiers. The retraining process involved retraining the classifier starting from different layers, namely Layers 1, 2, and 3, along with their preceding layers. Thus, the rows labeled Layer 1, Layer 2, and Layer 3 in Table 6-2 display the results from retraining each respective layer. For each starting layer a hyperparameter search for w was conducted as explained in Section 5-2-2.

Within Table 6-2, the italicized results represent the best-performing classifiers among all the retrained models for each individual starting layer where each result corresponds to the specific w_{best} value. The term “best-performing" refers to the classifier that achieved the lowest validation loss during retraining. Further, the non-italicized results in Table 6-2 indicate the classifiers that performed the best on the validation dataset by retraining them with a dataset resulting from the highest tested w value preceding w_{best}.

The findings from Table 6-2 demonstrate that the IHDA algorithm did not lead to any improvement in classification performance across any of the retrained layers. In fact, the performance of the retrained classifiers decreased compared to the original “20%-Classifier”. For instance, the accuracy declined from 85.39% to 80.51% for Layer 1, to 81.72% for Layer 2, and to 84.17% for Layer 3. Similar observations can be made in Table 6-2 for the validation loss and F1 score.

This performance drop of up to 4.88%, in case of Layer 1, seems undesirable, however, it becomes even more notable when considering that w_{best} represents the w value that yielded the fewest positive potential identified samples N_{pp} and, consequently, the smallest retrain dataset, with none of them exceeding a number of $\mathrm{N}_{pp} = 133$ positive potential samples. The quantities of positive potential samples for each hyperparameter search are detailed in Table 6-2 under the column N_{pp}.

Considering the concepts of Gradient Descent (GD) and Stochastic Gradient Descent (SGD) introduced in Section 2-3-2, where larger datasets divide into batches to calculate the network's gradient, one can infer that the larger the dataset, the more batches are used within a single training epoch. Hence, more parameter updates are executed in one epoch. As w_{best} corresponds to the smallest non-zero dataset in all cases, it appears that the best-performing classifiers are those where the IHDA algorithm made the least parameter updates.

This hypothesis gains more weight as classifiers retrained with a w value preceding w_{best} utilized a larger retrain dataset, which resulted in a larger performance drop of the retrained classifier. For example, Layer 1's accuracy decreased from 80.51% at $\mathrm{N}_{pp} = 133$ positive potential samples to 67.14% at $\mathrm{N}_{pp} = 11148$ samples, while Layer 2 witnessed a decrease from 81.72% at $\mathrm{N}_{pp} = 104$ samples to 58.39% at $\mathrm{N}_{pp} = 9148$ samples.

Interestingly, despite the rise in positive potential samples from $\mathrm{N}_{pp} = 110$ to $\mathrm{N}_{pp} = 9912$, Layer 3's performance drop was considerably less, with accuracy only decreasing from 84.17% to 82.51%. This could potentially be ascribed to Layer 3's position as the deepest layer in the network. When retraining Layer 3, parameters of Layers 1 and 2 remained unchanged, resulting in fewer overall network parameters being updated, specifically, only those defining Layer 3. This observation aligns with the hypothesis suggesting fewer parameters updated by the IHDA contributes to better preservation of the classifier's performance.

Further backing for this hypothesis can be drawn from the finding that peak performance was reached after a single retraining epoch, even though multiple epochs were performed. This holds true across all three layers and two different w values for each layer presented in Table 6-2, where for each retrained classifier the corresponding epochs are listed in the column Epoch.

To conclude, despite attempts, the results reported by Khan and Fraz [22] could not be replicated. Potential reasons could include missing hyperparameter specifications, such as the learning rate used, or the use of a different Variational Autoencoder (VAE) loss function than specified. Nonetheless, numerous instances in literature have successfully used VAEs for data augmentation, which consequently enhanced classifier performance [7, 27].

These findings underscore the need for further investigations to gain a deeper understanding of the factors contributing to the lack of improvement in classifier performance using the IHDA algorithm. Specifically, the performance of the employed VAEs in reconstructing input samples warrants evaluation. Given that these reconstructed samples are used for retraining the classifier, the performance of the VAEs that generate them significantly impacts the IHDA algorithm's effectiveness. Thus, the analysis will further entail an assessment of the reconstruction abilities of the utilized VAEs.

Model	**Loss**	**Accuracy [%]**	**F1 Score [%]**	**w**	**N$_{pp}$**	Epoch
20%-Classifier	0.4667	85.39	86.42	-	-	
Layer 1	*0.7313*	*80.51*	*82.01*	*0.5*	*133*	*1*
	2.154	65.46	67.14	0.45	11148	1
Layer 2	*0.5254*	*81.72*	*83.01*	*0.5*	*104*	*1*
	8.827	58.39	58.56	0.45	9148	1
Layer 3	*0.6260*	*84.17*	*85.39*	*0.5*	*110*	*1*
	1.870	82.51	83.82	0.45	9912	1

Table 6-2: Performance Metrics for Retrained Classifiers using the IHDA Algorithm: This table presents the validation loss, accuracy, F1 score, w values, quantity of positive potential samples (N$_{pp}$), and epochs for the original 20%-Classifier and retrained classifiers commencing from Layers 1, 2, and 3. Results highlighted in italics correspond to the models which delivered the lowest validation loss for each starting layer, each associated with a specific w_{best} value. Non-italicized results correspond to classifiers performing best on the validation dataset when retrained with a dataset resulting from the highest tested w value preceding w_{best}.

Assessment of Reconstruction Performance

The reconstruction performance of the employed VAEs is evaluated by generating the reconstructed validation dataset utilizing the VAE under evaluation. The performance of the "20%-Classifier" on the corresponding dataset is evaluated as it serves as a measure of the VAE's reconstruction capability, as motivated in Section 5-4.

The outcomes of these evaluations are summarized in Table 6-3. Here, the rows labeled "VAE 1", "VAE 2", and "VAE 3" are linked to the performance metrics of the "20%-Classifier" when evaluated on datasets produced by their respective VAEs. Subsets of these datasets were utilized by the IHDA algorithm to initiate retraining of the corresponding layer in the classifier.

Noteworthy is the similar performance of the reconstructed validation datasets from "VAE 1" and "VAE 3" in terms of accuracy and F1 score. For instance, the "20%-Classifier" yields an accuracy of 35.22% on the dataset reconstructed by "VAE 1" and 36.43% on the dataset reconstructed by "VAE 3". These values may appear low when compared to the accuracy of the original dataset, which scored 85.39% and served as the source for generating the reconstructed datasets.

Dataset	**Loss**	**Accuracy**[%]	**F1 Score**[%]
Original	0.1937	85.39	86.42
VAE 1	3.805	35.22	27.71
VAE 2	5.326	13.08	4.447
VAE 3	1.837	36.43	30.10

Table 6-3: Performance of the benchmark classifier on the original validation dataset and the datasets reconstructed by VAE 1, VAE 2 and VAE 3.

"VAE 2", however, performed considerably worse, yielding an accuracy of 13.08%, which is more than 20% less than "VAE 1" and "VAE 3". Aiming to comprehend why the validation dataset generated by "VAE 2" underperformed relative to those generated by "VAE 1" and

"VAE 3", the performance metrics of the different VAEs on the validation dataset were examined. These metrics were considered in terms of the Evidence Lower Bound (ELBO), Mean Square Error (MSE), and KL divergence and are listed in Table 6-4.

In assessing these metrics, particular interest is placed on the validation MSE and the validation KL divergence. "VAE 2" demonstrates a reconstruction MSE of $1.823 \cdot 10^{-2}$, similar to "VAE 1's" MSE of $2.003 \cdot 10^{-2}$. However, a difference appears in the KL divergence, where "VAE 1" shows a value of 1.653, while "VAE 2" is nearly zero at $2.980 \cdot 10^{-7}$. This near-zero KL divergence for "VAE 2" suggests that there is a high likelihood that most, if not all, of its latent variable distributions, have collapsed, which is undesirable in the application of the IHDA algorithm.

Model	ELBO	MSE	KL Divergence
VAE 1	958.35	0.02003	1.653
VAE 2	635.04	0.01823	$2.980 \cdot 10^{-7}$
VAE 3	645.19	0.08502	1.810

Table 6-4: Validation performance of VAE 1,VAE 2, and VAE 3 measured by the Evidence Lower Bound (ELBO), Mean Square Error (MSE), and Kullback-Leibler (KL) Divergence.

For further interpretations of these results, additional statistics are analyzed concerning the three different representations of the training dataset, which serve as the input to layers 1, 2, and 3 of the classifier and were employed for training the corresponding VAEs 1,2 and 3. The mean and standard deviation of these datasets are provided in Table 6-5.

Notably, the data used for training "VAE 2" exhibits the lowest mean of 0.03397 and the lowest standard deviation of 0.1410. By contrast, the data used for training VAE 1 displays a mean of 0.1745 and a standard deviation of 0.2675, with similar values reported for "VAE 3".

Given this disparity, the fact that"VAE 1" and "VAE 2" achieve similar MSE values on their validation dataset, as displayed in Table 6-4, might suggest that "VAE 1" fits the data better than "VAE 2". This assertion comes from the understanding that the training data for "VAE 2" is numerically closer to zero than that for "VAE 1", as inferred from Table 6-5. Therefore, achieving the same MSE could mean a relatively worse fit for "VAE 2". This implies that for "VAE 2" to preserve the same level of distinguishing features in its reconstructed samples as "VAE 1", it should achieve a lower MSE.

Model	Mean	Standard Deviation
VAE 1	0.1745	0.2675
VAE 2	0.03397	0.1410
VAE 3	0.1573	0.3946

Table 6-5: Mean and standard deviation of dataset used to train VAE 1, VAE 2, and VAE 3.

To summarize, it was observed that the validation datasets reconstructed by "VAE 1", "VAE 2", and "VAE 3" perform differently on the "20%-Classifier", with "VAE 2's" data showing the lowest performance. Furthermore, the training data for "VAE 1", "VAE 2", and "VAE 3" were found to have distinct characteristics. Specifically, the data used to train "VAE 2" is numerically closer to zero than those for "VAE 1" and "VAE 3".

All three VAEs were trained with the same loss function, which was introduced by (3-17) and assumes an equal weighting between the MSE and KL terms. Taking into account these observations, it can be inferred that the lower performance of "VAE 2's" reconstructed data might be due to the assumption of equal weight for the VAE and KL divergence term in the loss function. Specifically, "VAE 2" might necessitate a higher weight on the MSE term to generate a reconstructed training dataset that performs as well on the "20%-Classifier" as those generated by "VAE 1" and "VAE 3".

These findings emphasize the importance of adjusting the weighting of the MSE and KL divergence in the loss function based on the specific characteristics of the training datasets, which underscores the motivation for the first proposed enhancement to the IHDA method, as introduced and discussed in Section 5-3-1:

> **Proposal 1:** Optimizing the Relative Contributions of the Kullback-Leibler (KL) Divergence and Mean Square Error (MSE) Terms in the VAE Loss Function.

Due time and computation constraints, discussed in Section 5-2-4, only VAE 1 will be used for retraining the layers of the classifier moving forward.

6-3 Exploring the Results of Proposal 1

6-3-1 Performance Evaluation: β-VAE and σ-VAE

As discussed in Section 5-3-1, one aspect of evaluating Proposal 1 involves a comprehensive analysis of σ-VAE's performance relative to β-VAE. For this performance assessment, several β-VAEs defined by the β values $[0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1, 5]$ were trained.

Assessing the Evaluation Metrics for β-VAE and σ-VAE

The first assessment of the trained β-VAEs and the σ-VAE is based on the VAE's validation metrics: ELBO, MSE, and Kullback-Leibler divergence. These metrics are depicted in Figure 6-2, where the β-VAE performance is marked by blue dots and the σ-VAE performance is indicated by an orange star.

Upon first inspection of Figure 6-2 it appears that σ-VAE achieves a balance between MSE and KL divergence values, operating in a region where neither metric reaches extreme values compared to the tested β-VAEs. Further, it should be observed that σ-VAE achieves the lowest ELBO value, aligning with its intended design detailed in Section 3-3-1. However, this does not translate to achieving the lowest values for MSE or KL divergence.

An increase in the β value within the β-VAE implies a higher emphasis on the KL divergence term compared to the MSE term in the VAE loss function. The optimization of this modified loss function consequently results in a reduction in the KL divergence value and an increase in the MSE of the resultant VAE. This inverse relationship between the MSE and the KL divergence can be observed in Figure 6-2.

Thus, a lower MSE value than that of the σ-VAE can be accomplished by choosing a β, value lower than the one associated with σ-VAE. In the context of the IHDA algorithm, which

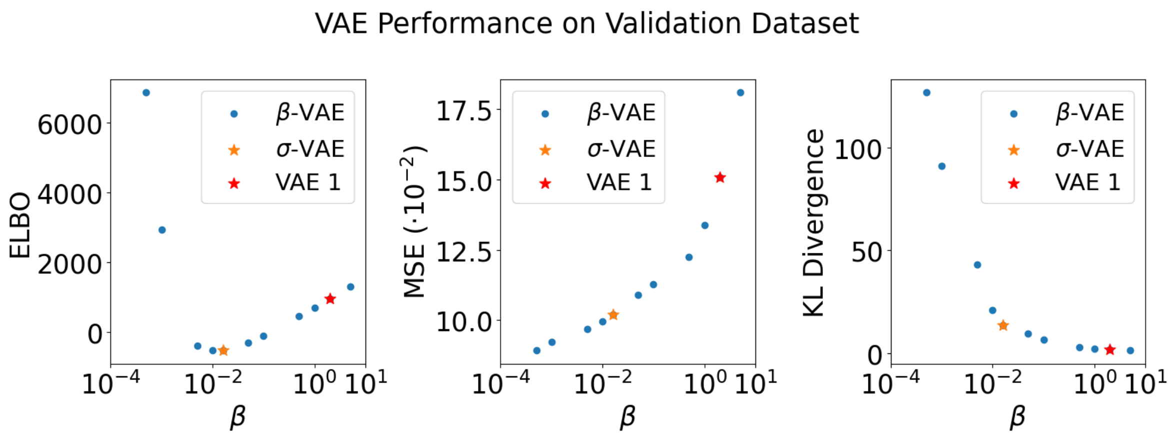

Figure 6-2: Comparison of ELBO, MSE, and KL divergence for different variants of β-VAE, along with performance of σ-VAE and VAE 1. The blue dots signify the β-VAE performances, while the orange star represents the performance of σ-VAE and the red star represents the performance of VAE 1.

employs the VAE for data generation, a smaller MSE might seem advantageous at first. However, as depicted in Figure 6-2, beyond a certain β value, such a decision leads to a rapid increase of KL divergence, indicating an increasing discrepancy between the distributions of latent variables and the assumed standard Gaussian prior. As expanded upon in Section 5-3-1, beyond a certain threshold, this escalation in KL divergence will have detrimental effects on the performance of the IHDA algorithm.

Since the precise KL divergence threshold impacting the IHDA algorithm's performance is not clearly defined, it is plausible that certain β values could yield β-VAEs models that, despite exhibiting a higher KL divergence, demonstrate a lower MSE than the σ-VAE and may be more suited for the IHDA algorithm. However, finding this β value requires extensive tuning.

As the σ-VAE operates within a region where neither the MSE nor the KL divergence reaches extremes, it appears to be a suitable, but not necessarily the optimal choice for the IHDA algorithm. However, since MSE does offer limited insights into the VAE's ability to retain distinguishing features, as outlined in Section 5-4, the subsequent evaluation will focus on this aspect.

Evaluating Reconstruction Performance: β-VAE versus σ-VAE

The trained VAE's capacity to maintain distinguishing features is assessed by benchmarking the "20%-Classifier" on the reconstructed validation dataset. As discussed in Section 5-4, this process offers a quantitative measure of the VAE's ability in preserving distinctive features and facilitates a more comprehensive understanding of the performance of σ-VAE in comparison to β-VAE.

Figure 6-3 illustrates the comparison between the performance of the "20%-Classifier" evaluated on the validation dataset reconstructed by various configurations of β-VAE and σ-VAE. It demonstrates that higher β values, in comparison to the β value associated with σ-VAE, lead to a decrease in performance across all three metrics: loss, accuracy, and F1 score. This observation is consistent with the findings depicted in Figure 6-2, where an increase in β values resulted in an increase in MSE.

Further analysis of the σ-VAE's capacity to retain distinguishing features, compared to the β-VAE, shows that while the highest tested β value achieved less than 30% accuracy, the σ-VAE achieved over 50% accuracy. The best-performing tested β value achieved an accuracy exceeding 60%, positioning the σ-VAE closer to the best performing of the tested β-VAE configurations rather than the poorest performer.

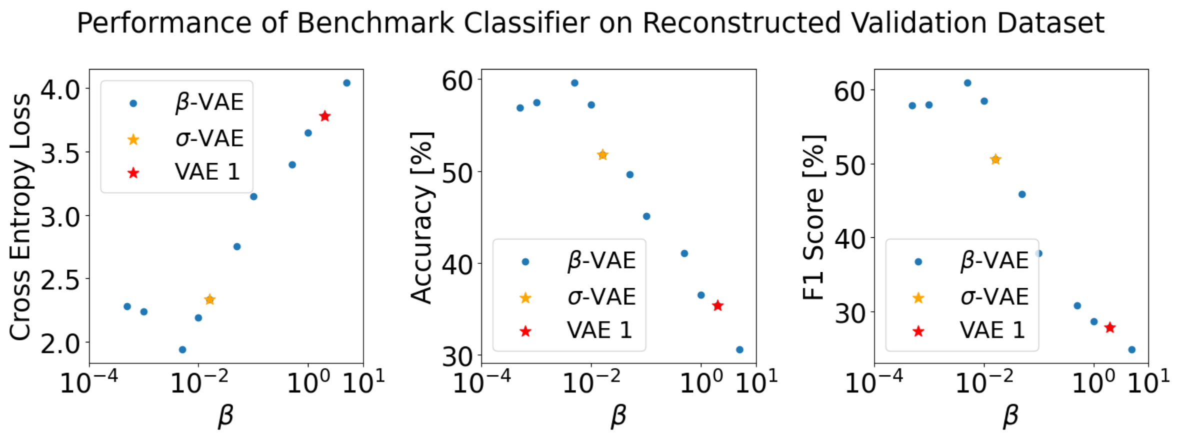

Figure 6-3: Performance of the "20%-Classifier" on the validation dataset reconstructed by various configurations of β-VAE, σ-VAE and VAE 1.

Integrating the findings from Figure 6-2 and Figure 6-3 uncovers an important nuance. While for the tested β values, the MSE consistently decreases as the β values decrease the performance of the benchmark classifier on the reconstructed dataset does not consistently increase. In fact, in the considered case, at some point, the performance starts to even decrease for smaller MSE. This suggests the presence of an optimal β value, where the classifier achieves the best performance on the corresponding reconstructed dataset, as indicated by the lowest validation loss and highest accuracy and F1 score.

Model	ELBO	MSE	KL Divergence
β-VAE with $\beta = 0.001$	-301.82	0.01261	45.73
σ-VAE	-538.12	0.01356	13.58

Table 6-6: Comparison of the validation performance metrics including ELBO, MSE, and KL divergence for trained σ-VAE and β-VAE with $\beta = 0.001$.

Among the tested β values, a β value of 0.001 seems to be a preferable choice for the IHDA algorithm over the σ-VAE. As inferred from Table 6-7, the "20%-Classifier" yields an accuracy of 60.12% on the dataset reconstructed by β-VAE versus 51.88% for σ-VAE, indicating a better capability in generating samples with distinguishing features for β-VAE. The increase in KL divergence, listed in Table 6-6, from 13.58 for σ-VAE to 45.73 for the considered β-VAE, does not appear substantial, given that a 64-dimensional latent space is considered, and the total KL divergence of the VAE represents the sum of the KL-divergence values for each latent variable distribution. Therefore, this observation reinforces the statement made in the previous section that there may exist certain β-VAE that could potentially serve better for the IHDA algorithm, however, determining these β values will demand manual tuning.

Dataset	**Loss**	**Accuracy** [%]	**F1 Score**[%]
β-VAE (with $\beta = 0.001$)	1.901	60.12	61.01
σ-VAE	2.334	51.88	50.68

Table 6-7: Performance of the "20%-Classifier" on datasets reconstructed by β-VAE with $\beta = 0.001$ and σ-VAE.

Conclusion of the comparison β-VAE and σ-VAE:

While β-VAE permits adjusting the VAE's characteristics, potentially resulting in a model better suited for its intended use, the σ-VAE operates in a balanced region where neither the KL divergence nor the MSE reaches extreme values, which aligns with the requirements of the IHDA algorithm. Moreover, the performance difference between the best performing tested β-VAE and σ-VAE in preserving distinguishing features seems to be tolerable, considering the latter's advantage of not requiring manual tuning. Thus, it is worthwhile to explore how σ-VAE performs within the IHDA algorithm.

6-3-2 Performance Evaluation: VAE 1 and σ-VAE

Before integrating σ-VAE in the IHDA algorithm, a comparative analysis between σ-VAE and the VAE utilized by Khan and Fraz [22] will be performed. The analysis will focus on evaluating the reconstruction performance of the two VAEs and their ability to map their input samples to a meaningful latent space that contains valuable information about the input samples. This analysis aims to provide additional insight into the utilized VAEs, which help to interpret and analysis of the results of implementing them in the IHDA algorithm.

Reconstruction Performance

As in previous Sections, to compare the ability of VAE 1 and σ-VAE to preserve distinguishing features in their reconstructed samples, the performance of "20%-Classifier" on the reconstructed validation dataset reconstructed by the considered VAEs is evaluated. The evaluation results are listed in Table 6-8.

From these results, the σ-VAE exhibits improved reconstruction performance. Accuracy and F1 score of the reconstructed datasets increased from 35.26% and 27.71%, respectively, for VAE 1 to 51.88% and 50.68% for σ-VAE. This indicates that σ-VAE outperforms the VAE proposed by Khan and Fraz [22] in terms of generating samples that contain distinguishing features, which justifies implementing σ-VAE in the IHDA algorithm.

Dataset	**Loss**	**Accuracy** [%]	**F1 Score**[%]
Original	0.1937	85.39	86.42
VAE 1	3.803	35.26	27.71
σ-VAE	2.334	51.88	50.68

Table 6-8: Performance of the "20%-Classifier" on original and reconstructed validation datasets generated by VAE 1 and σ-VAE. Metrics include cross entropy loss, accuracy, and F1 score for the original, VAE 1, and σ-VAE datasets.

For a more thorough understanding, representative samples from each class within the dataset were visually examined. In particular, Figures 6-4, 6-5, 6-7 and 6-8 visually present the original and reconstructed outputs yielded by both VAE 1 and σ-VAE. However, caution should be exercised when interpreting these results, as only a single sample from each class is considered, thus potentially introducing bias into the evaluation.

A shared challenge for both σ-VAE and VAE 1 is the difficulty in preserving high-frequency components of the input samples in their outputs. This could be due to the fact that the decoder in VAE represents a distribution, from which only the mean is used to reconstruct the input signal as highlighted in Section 3-2-4. While including the decoder variance could potentially enhance visual reconstruction quality, it is important to note that these high-frequency components may not be crucial for the correct classification of the samples.

Examining the samples reconstructed by VAE 1 and σ-VAE, it seems that σ-VAE better approximates the mean of the original data, especially for samples in the "Bike", "Walk", and "Run" classes. An instance of this is the accelerometer features of the "Walk" class sample shown in Figure 6-4. Here, the sample reconstructed by VAE 1 significantly deviates from its original sample, while the σ-VAE sample mirrors the original data more closely.

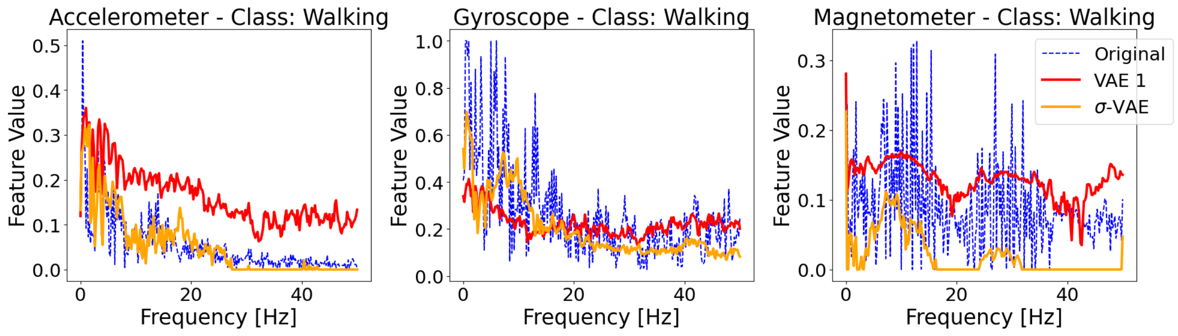

Figure 6-4: A visual comparison of the original and reconstructed samples of the "Walk" class generated by VAE 1 and σ-VAE.

Interestingly, both VAE models map the accelerometer and gyroscope features of the motorized transportation modes "Bus", "Train", and "Subway", as well as the "Still" mode, which is shown in Figure 6-5, to zero in their reconstructed samples, despite the original non-zero values. This indicates a loss of feature information for these transportation modes in both VAE 1 and σ-VAE. In contrast, less feature loss is observed for the "Bike", "Run", and "Walk" modes, as their accelerometer and gyroscope data is not mapped to zero.

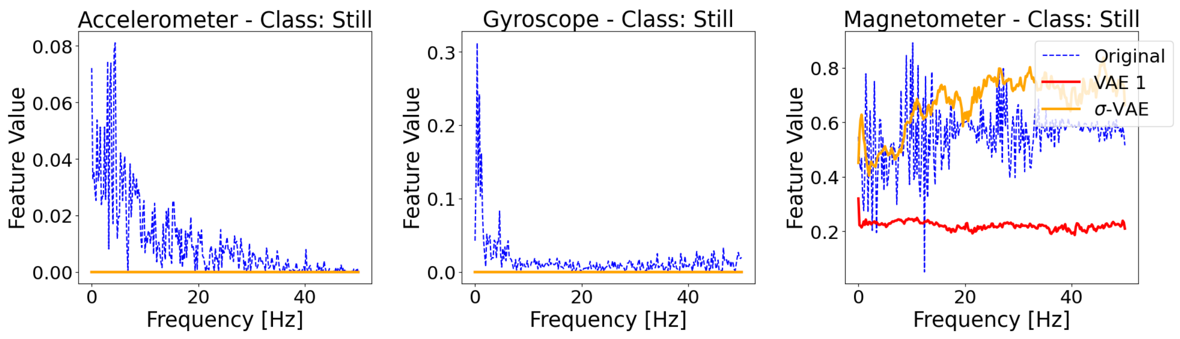

Figure 6-5: A visual comparison of the original and reconstructed samples of the "Still" class generated by VAE 1 and σ-VAE.

The underperformance of the VAEs in reconstructing the accelerometer and gyroscope features of the motorized transportation modes and "Still", may originate from the applied feature normalization process outlined in Section 5-1-2. This process scales features along the "feature" axis rather than the "sample" axis, employing the magnitudes of the Fourier transformations of accelerometer, gyroscope, and magnetometer data.

In terms of the original time-domain data, the accelerometer and gyroscope readings potentially exhibit higher magnitudes for the "Run", "Bike", and "Walk" modes, due to more intense physical movement. This in turn leads to higher magnitudes in the Fourier domain for these modes. Consequently, when the inter-feature normalization is applied, these higher-magnitude features get mapped to higher values than those of motorized transportation modes, including "Still".

The VAE loss function incorporates an MSE component that disproportionately penalizes larger deviations between the original and reconstructed samples. When the features are on different scales, as in this case, this can introduce a bias during the optimization of the VAE loss function. More specifically, the model may pay more attention to minimizing deviations in the features of the "Run", "Bike", and "Walk" modes that have larger values after normalization. This could result in the VAEs performing less effectively in reconstructing samples for motorized transportation modes and "Still", where the corresponding features typically map to smaller values.

This loss in feature representation could partially explain why the "20%-Classifier", when evaluated on data reconstructed by σ-VAE, achieves an accuracy that is over 30% lower than that of the original dataset, as illustrated in Table 6-8.

To summarize, this section demonstrates that the σ-VAE outperforms the VAE in preserving distinguishing features, thus validating its inclusion in the IHDA algorithm. However, it also underscores a potential limitation in the reconstruction ability of the VAE, where due to a combination of the nature of data and the feature extraction process, particularly feature normalization, the VAEs show a preference towards better reconstructing accelerometer and gyroscope features for "Walk", "Run", and "Bike" modes, but perform less effectively for other modes.

Illustrating VAEs' Potential for Data Augmentation

To illustrate the potential of VAEs in generating multiple distinct but similar samples from a single sample, a brief excursion is warranted. Displayed in Figure 6-6 are five variations of the "Walk" sample reconstructed by the σ-VAE. The original sample was initially depicted in Figure 6-4. The process of generating those samples involves mapping the original sample to its latent distributions, sampling latent space representations from these distributions, and then decoding them back to the observation space. As observed, all five generated samples exhibit similarities, indicating that they are variations of the original sample. However, they also demonstrate subtle differences, highlighting the Data Augmentation (DA) capabilities of VAEs.

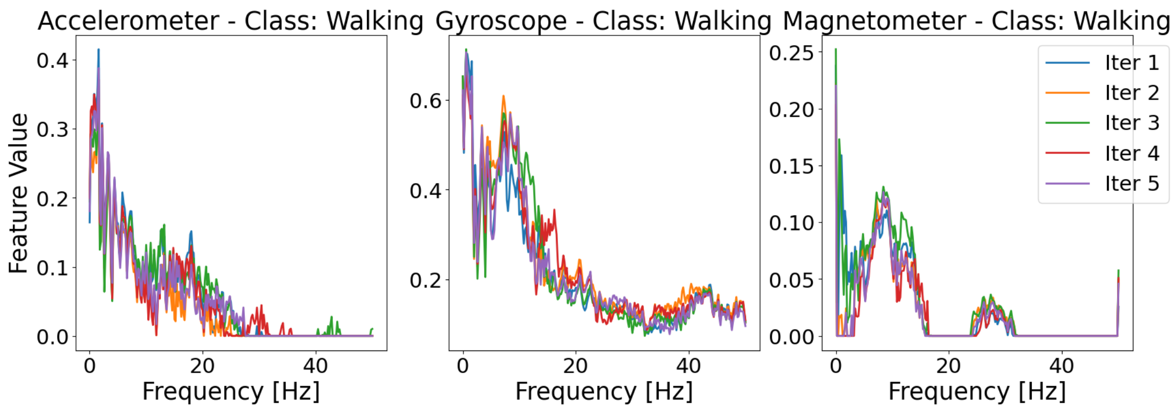

Figure 6-6: Five reconstructed samples generated by σ-VAE from a single "Walk" sample. The original sample is represented in Figure 6-4.

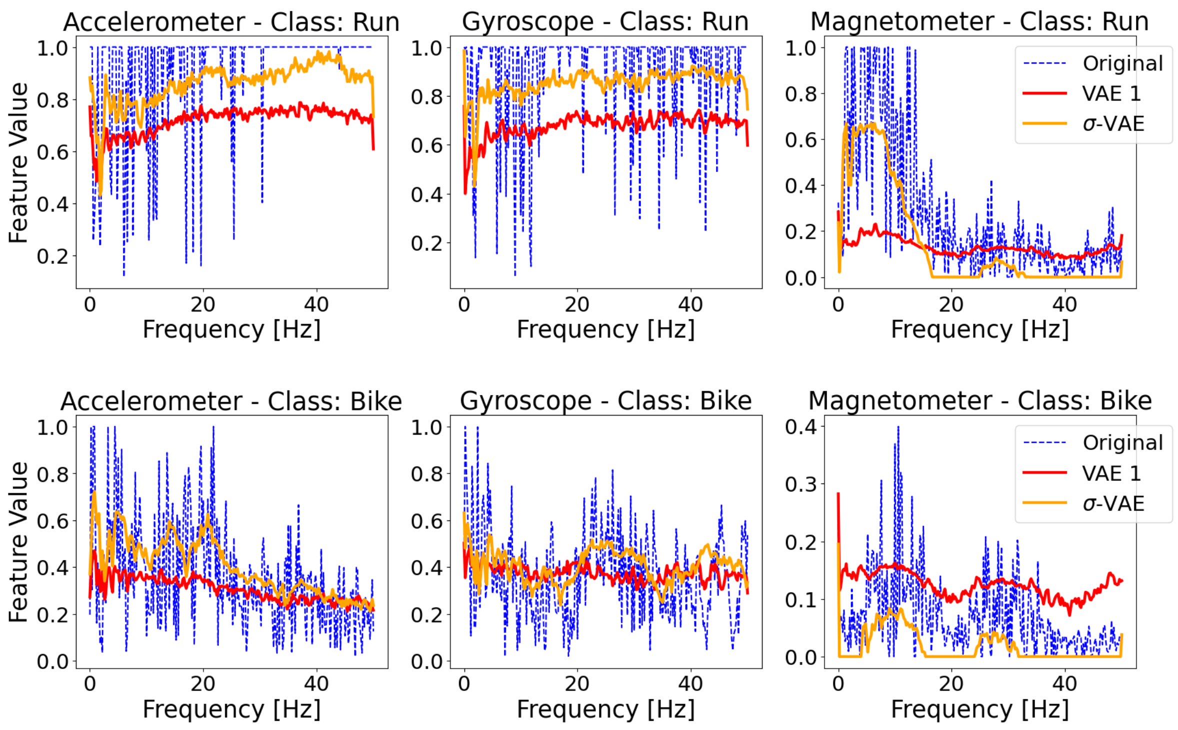

Figure 6-7: A visual comparison of the original and reconstructed samples of the "Run" and "Bike" classes generated by VAE 1 and σ-VAE.

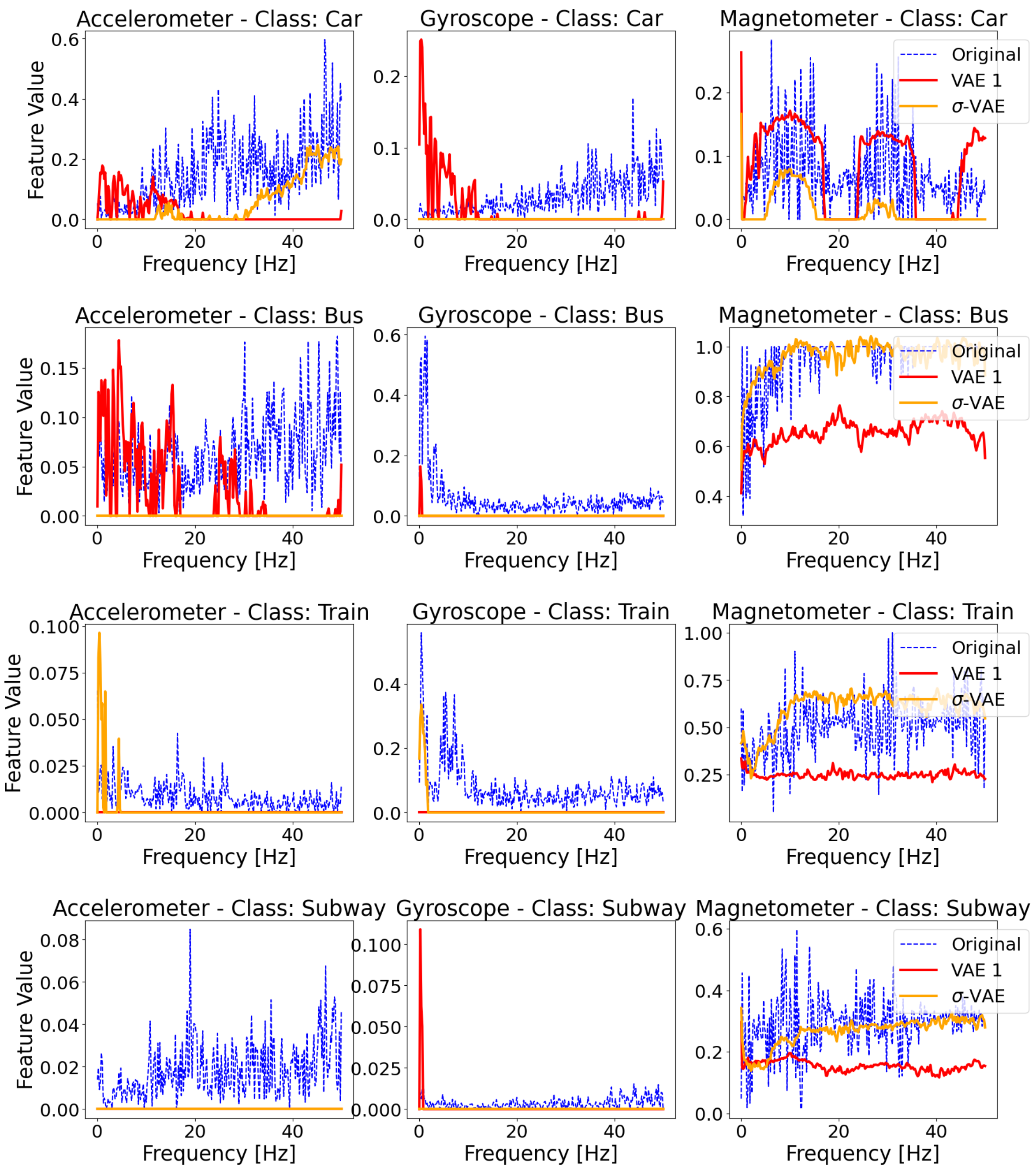

Figure 6-8: A visual comparison of the original and reconstructed samples of the “Car”, “Bus”, “Train” and “Subway” classes generated by VAE 1 and σ-VAE.

Latent Space

The focus now shifts back to the comparative evaluation of the σ-VAE and the VAE proposed by Khan and Fraz [22], referred to as VAE 1.

The IHDA algorithm determines the similarity between samples by calculating the cosine similarity of their respective latent vectors. The aim of the following evaluation is to ascertain whether using cosine similarity in conjunction with the σ-VAE enhances the process of measuring similarity between input samples based on their corresponding latent vectors in comparison to combining cosine similarity metrics with the VAE proposed by Khan and Fraz [22].

Table 6-9 presents a comparative analysis of the KL divergence for both VAE models on the validation set. Interestingly, the KL divergence increased from 1.653 for VAE 1 to 13.58 for σ-VAE. Given a latent space dimension of 64, the KL divergence value of 1.653 attained by VAE 1 appears relatively low and suggests that multiple latent variable distributions may face posterior collapse. Thus, the elevated KL divergence value achieved by σ-VAE could indicate a reduced degree of posterior collapse among its latent variables. In turn, this suggests that σ-VAE is more effective in preserving information in its latent space, enhancing the comparison of input samples based on their latent vectors using the cosine similarity metric.

To further strengthen this premise a supplementary visual inspection is introduced.

Model	ELBO	MSE	KL Divergence
VAE 1	958.35	0.02003	1.653
σ-VAE	-538.12	0.01356	13.58

Table 6-9: Comparative analysis of the validation performance of VAE 1 and σ-VAE. The metrics reported include the Evidence Lower Bound , MSE, and KL divergence.

Figures 6-10 and 6-11 display the distribution of cosine similarities between a randomly chosen sample of the “Bike” class and samples from the same class, represented by the blue distribution, as well as the similarities to samples from different classes, depicted by the orange distribution.

The theoretical expectation is that samples belonging to the same class as the considered sample would, on average, exhibit higher cosine similarity compared to samples from different classes. This expectation arises from the assumption that samples belonging to the same class are inherently more similar to the considered sample than samples from different classes.

However, in the context of VAE 1, an examination of the figures contradicts this expectation. Consider, for instance, Figure 6-9, which illustrates the distribution of cosine similarities for samples within the same class and those from the “Walk” class. It appears that the cosine similarity distribution for same-class samples mirrors that for “Walk” class samples, with a near-identical overlap between the two distributions. This pattern, noticeable across all classes, points to a substantial loss of meaningful latent variable information in the latent space, thereby suggesting a strong likelihood of posterior collapse.

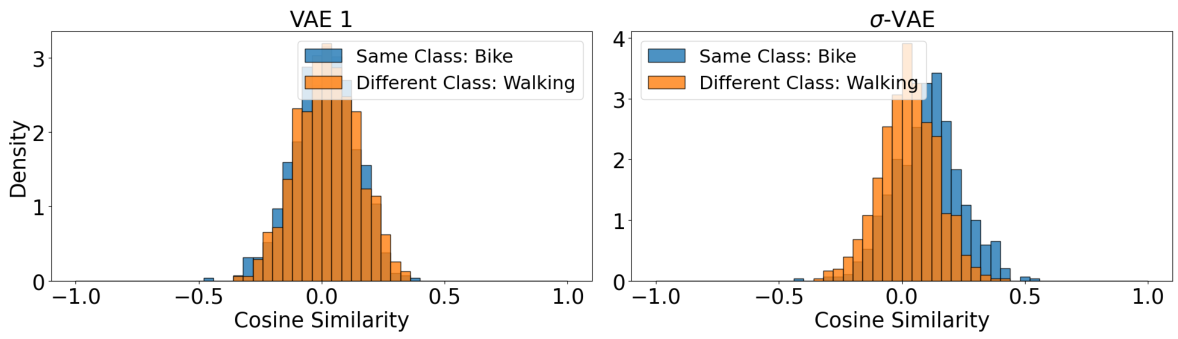

Figure 6-9: Histogram of cosine similarities between a randomly drawn sample of class "Bike" and samples from the same class and the cosine similarity between the considered sample and samples of class "Walk" for VAE 1 and σ-VAE.

On the other hand, examination of the histograms for σ-VAE reveals a notable improvement. The distribution of distances to samples from the same class demonstrates higher cosine similarity values than the distribution for samples from different classes. This differentiation creates regions with minimal or no overlap between the distributions, signaling a more distinct clustering of classes in the latent space and suggesting a lower degree of posterior collapse compared to VAE 1.

The analysis further reveals that for the histograms belonging to σ-VAE, the region with no overlap is smallest for the transportation modes "Run" and "Walk", while it is larger for motorized modes including the "Still" mode. This pattern demonstrates that the "Bike" class sample is computed as being less similar to motorized transportation modes, including "Still", than it is to "Run" and "Walk". This finding aligns with prior research that emphasized a distinct separation between active ("Run", "Bike", "Walk") and passive ("Bus", "Car", "Subway", "Train", "Still") modes, attributed to the considerably different motion patterns inherent to active and passive modes [41, 43].

This concrete example underscores that comparisons based on latent vectors mirror real-world observations, suggesting that the latent space encapsulates meaningful information about the input samples and that similarities/dissimilarities between input samples can be quantified using the cosine similarity of corresponding latent space representations.

To summarize, the latent space of the σ-VAE has been observed to result, on average, in greater cosine similarities between samples from the same class than between samples from different classes, a characteristic not seen with VAE 1. This feature, coupled with the enhanced KL divergence demonstrated by the σ-VAE, indicates a lower degree of posterior collapse in the σ-VAE as opposed to VAE 1, which supports the application of σ-VAE in the context of the IHDA framework.

6-3-3 Summary of Results Proposal 1

Summarizing the findings from Section 6-3-2, the σ-VAE provides a generalizable strategy for optimizing the balance between the KL divergence and the MSE within the VAE loss function. Although the β-VAE may demonstrate improved performance under specific conditions, it necessitates manual tuning which the σ-VAE circumvents. Further, σ-VAE has evidenced considerable improvements over the VAE model proposed by Khan and Fraz [22]. These enhancements are attributed to the σ-VAE's augmented ability to retain distinctive features in its generated samples and its enhanced proficiency for mapping input data to a meaningful latent space.

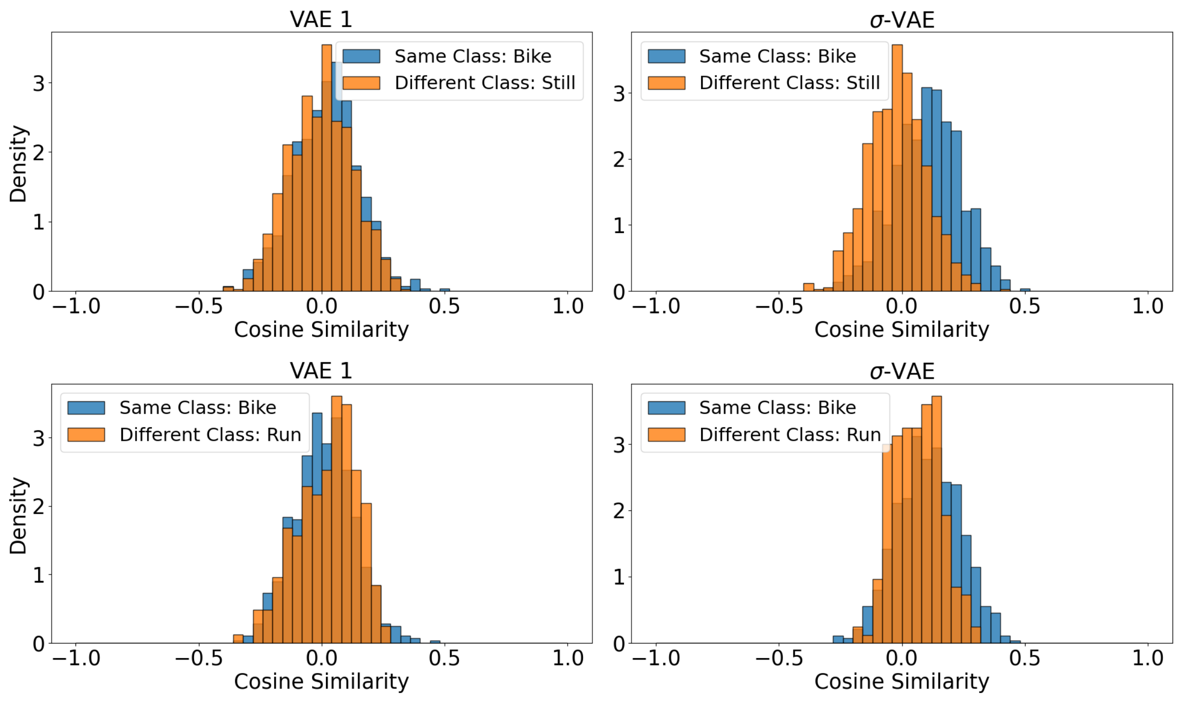

Figure 6-10: Histogram of cosine similarities between a randomly drawn sample of class "Bike" and samples from the same class and the cosine similarity between the considered sample and samples of class "Still" and "Run" for VAE 1 and σ-VAE.

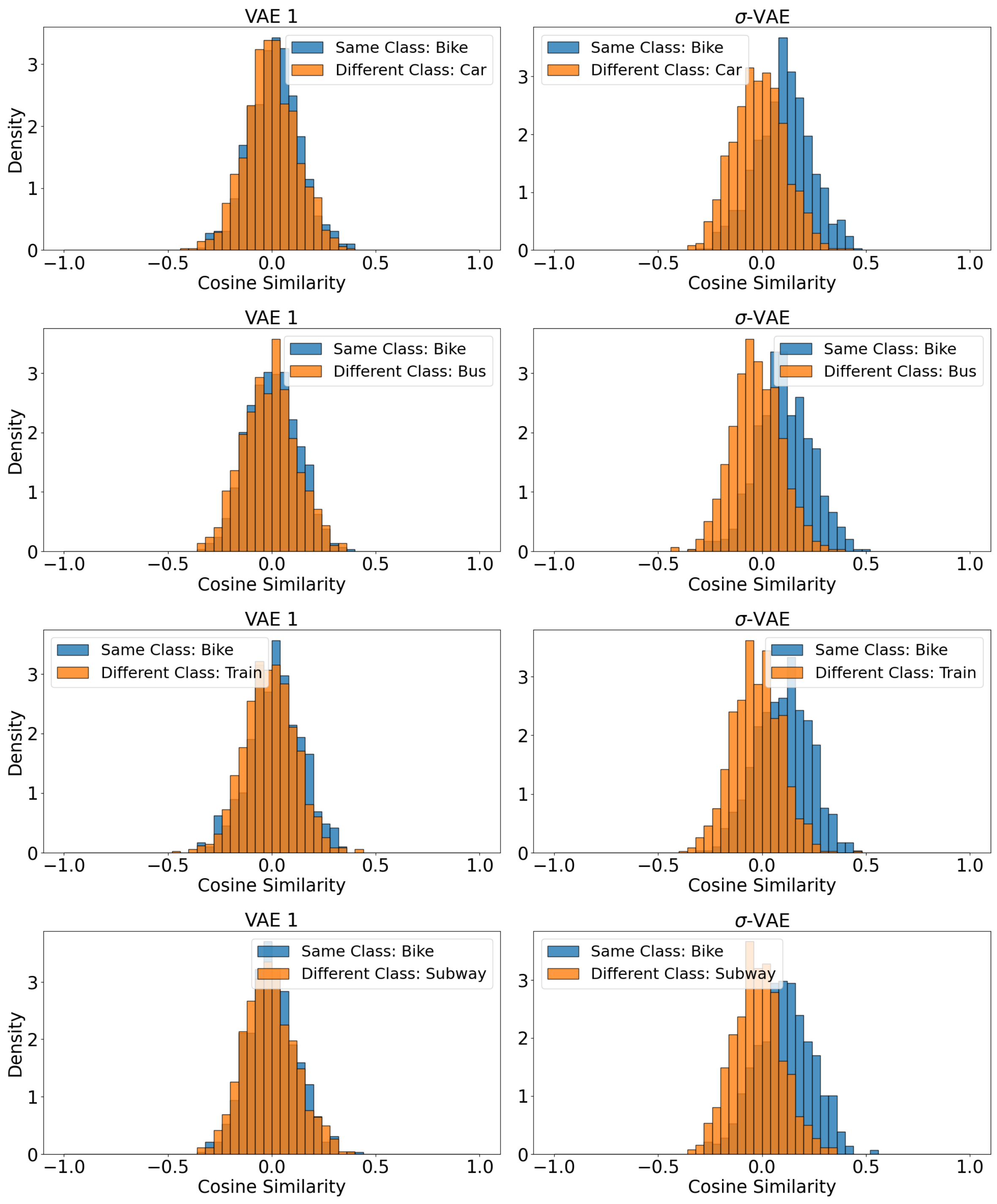

Figure 6-11: Histogram of cosine similarities between a randomly drawn sample of class "Bike" and samples from the same class and the cosine similarity between the considered sample and samples of class “Car”, “Bus”, “Train” and “Subway” for VAE 1 and σ-VAE.

6-4 Results Proposal 2

Although in the previous section it was indicated, that σ-VAE has decreased the degree of facing posterior collapse compared to the VAE proposed by Khan and Fraz [22], it is important to acknowledge that posterior collapse may still occur in certain latent variables of σ-VAE.

To further enhance the IHDA algorithm's capability to assess sample similarity based on their latent space distributions, a second proposal has been put forward. As discussed in detail in Section 5-3-2, this proposal focuses on:

> **Proposal 2:** Integrating the KL Divergence as a Reliable Latent Space Similarity Metric in the IHDA Algorithm.

Employing KL-Divergence as a Distance Metric

In this section, the use of KL divergence as an alternative distance metric to the cosine similarity will be evaluated, while still utilizing σ-VAE. Similar to the previous section, a visual inspection will be conducted to compare the results obtained using the two different distance metrics.

Figures 6-13 and 6-14 display distributions similar to those in the preceding section. These figures depict the similarities between the same randomly selected sample from the "Bike" class, its distance to samples of the same class, and its distance to samples of a different class. The left-hand side plots utilize the cosine similarity metric, while the right-hand side employs KL divergence. Furthermore, the logarithm of KL divergence values is utilized to account for the nonlinear growth of the distance metric as the distance increases.

Understanding these figures requires acknowledging that an increase in KL divergence signifies an increase in the divergence between the two compared latent space distributions, contrary to the behavior of cosine similarity where an increase indicates a decrease in similarity. Therefore, for the KL divergence, samples from the same class as the selected sample are expected to display lower KL divergences on average compared to samples from different classes, contrary to the expectations with cosine similarity.

The visual inspection suggests a reduction in the overlap between the two plotted distance distributions when using KL divergence, indicating a stronger distinction between samples of the same class and samples of different classes. Thus, while the cosine similarity metric presented some insight into the similarity of two input representations based on their σ-VAE latent space representations, the KL-divergence appears to enhance this capability.

Interestingly, the previous section revealed that the cosine similarity metric, when combined with σ-VAE, yielded similar values for "Walk" and "Run", but the KL divergence shows notable improvement across all classes, and particularly for "Walk" and "Run". For instance, as demonstrated in Figure 6-12 for "Run", the distributions shifted from near total overlap with cosine similarity to almost zero overlap with KL divergence, indicating a clear enhancement in differentiation.

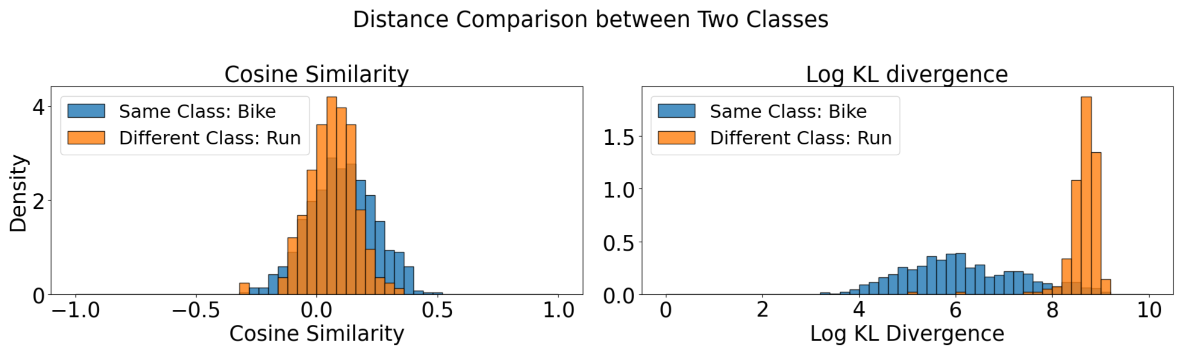

Figure 6-12: Histogram of cosine similarities and KL divergences between a randomly drawn sample of class “Bike” and samples from the same class and the cosine similarity between the considered sample and samples of class “Run” for σ-VAE.

These improvements infer that the KL divergence provides a more accurate evaluation of latent space representations, better capturing the similarity of input samples, possibly owing to KL divergence’s resilience to the impact of posterior collapse. Consequently, it appears that KL divergence offers a more extensive extraction of information about input samples stored in the latent space of σ-VAE compared to the cosine similarity metric.

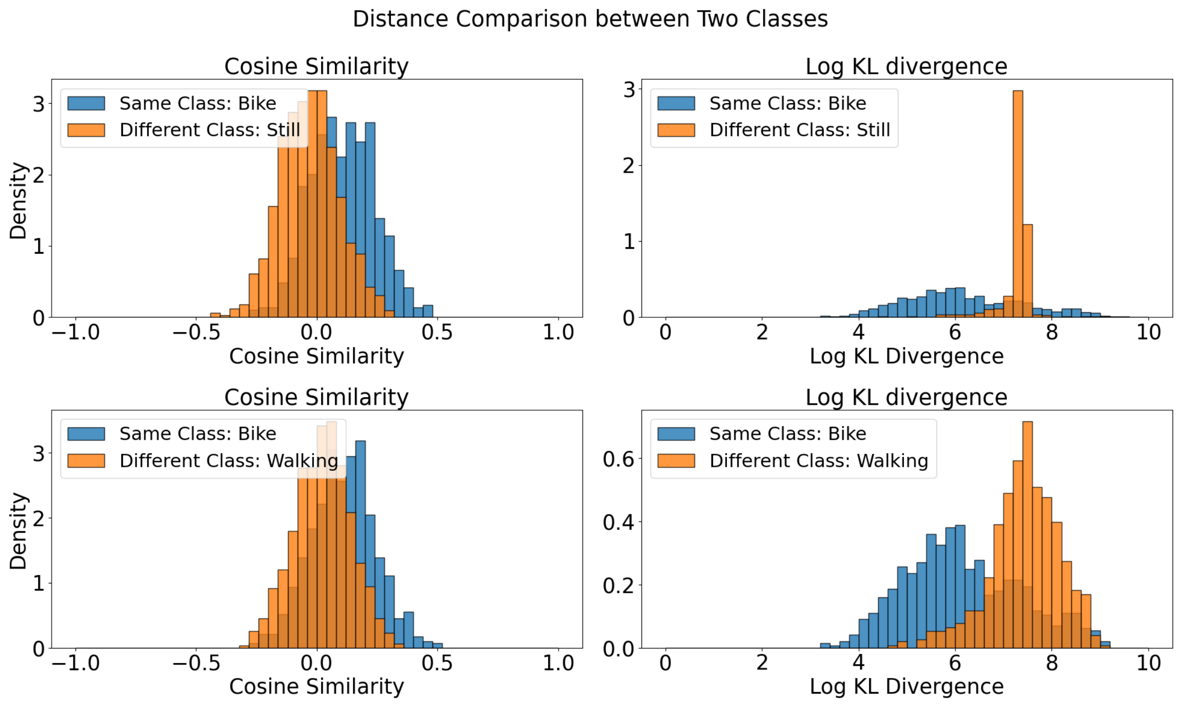

Figure 6-13: Histogram of cosine similarities and KL divergences between a randomly drawn sample of class “Bike” and samples from the same class and the cosine similarity between the considered sample and samples of class “Still” and “Walk” for σ-VAE.

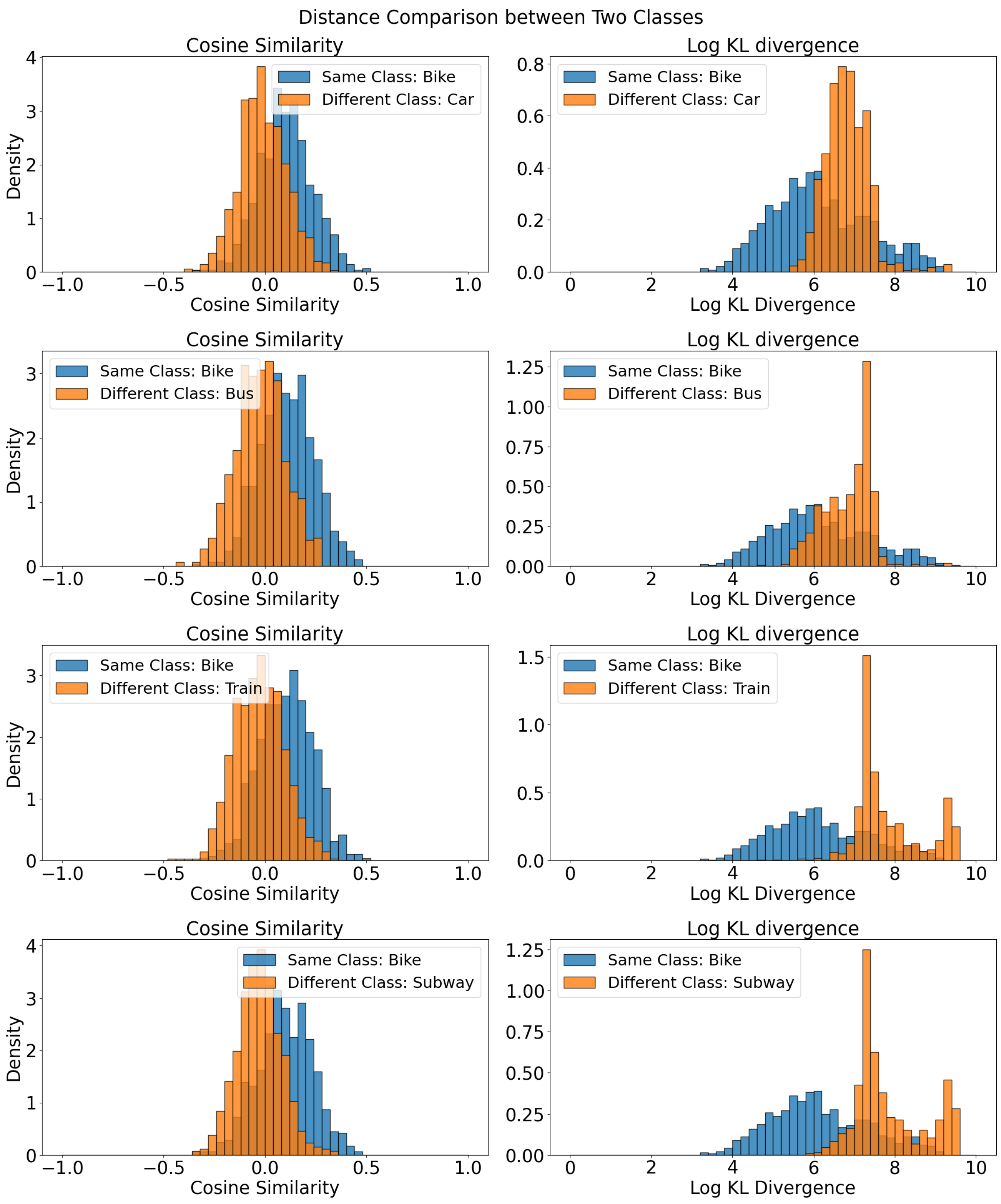

Figure 6-14: Histogram of cosine similarities and KL divergences between a random drawn sample of class “Bike” and samples from the same class and the cosine similarity between the considered sample and samples of class “Car”, “Bus”, “Train” and “Subway” for σ-VAE.

6-5 Evaluating the Identification Improvement

Before delving into the results obtained from integrating the proposed enhancements to the IHDA algorithm, this section evaluates how the enhancements presented in Section 6-3 and Section 6-4 contribute to the improved identification of potential positive samples in the IHDA algorithm. These identified samples play a critical role in retraining the classifier, thus directly influencing the performance of the algorithm. Therefore, an investigation of this identification process is warranted.

From a theoretical perspective, samples that may require additional sampling, which could improve the classifier's performance, are those that, on average, perform worse on the classifier than the entire dataset. To compare the performance of different configurations of VAEs and distance metrics in identifying positive potential samples, the "20%-Classifier's" performance on the identified positive potential train samples is evaluated for each tested w. Therefore, a lower classifier performance on the identified positive potential samples will serve as an indication of the improved performance of the IHDA algorithm in identifying positive potential samples.

Figure 6-15 presents the performance results of the "20%-Classifier's" on the identified positive potential dataset using VAE 1 and σ-VAE each combined with the cosine similarity metric in the IHDA algorithm, while Figure 6-16 focuses on the performance of the IHDA algorithm utilizing σ-VAE with the KL-divergence metric.

Examining Figure 6-15, when identifying positive potential samples using VAE 1, the loss function remains relatively constant until $w = 0.35$, after which it slightly decreases, resulting in an increase in accuracy. However, with the largest tested w of 0.5, the loss function increases again. This behavior deviates from the expected pattern for identifying positive potential samples.

In contrast, σ-VAE shows improvement in this regard. The loss function value starts at approximately 0.2 and begins to increase at $w = 0.15$, reaching over 0.3 for $w = 0.55$. As a result, the accuracy decreases to below 84% compared to the accuracy of the entire training dataset, which is 92.5%. This clear decrease in accuracy indicates a notable improvement in identifying samples that may require retraining. Based on this observation, utilizing σ-VAE instead of VAE 1 in the IHDA algorithm seems to enhance the algorithm's performance in identifying positive potential samples.

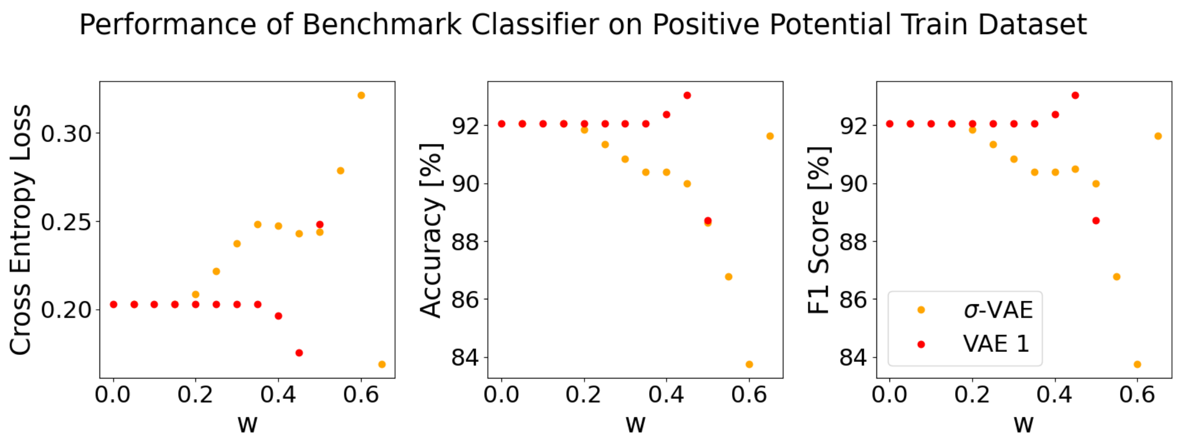

Figure 6-15: Performance results of the "20%-Classifier" on the identified positive potential dataset using VAE 1 and σ-VAE combined with cosine similarity.

Moving on to Figure 6-16, as the value of w decreases, the loss function consistently increases. The loss function for the identified positive potential dataset exceeds 0.5 when employing the KL divergence metric, while it reaches only 0.3 with cosine similarity metrics. Consequently, the accuracy decreases from 92.5% for the entire dataset to below 74% for the positive potential dataset identified using the KL divergence as a similarity metric, compared to an accuracy drop to 84% with cosine similarity metrics. This drop in accuracy suggests that utilizing the KL divergence as a similarity metric further enhances the algorithm's ability to identify underperforming samples.

This section thus confirms that the applied modifications enhance the IHDA algorithm's ability to identify samples on which the classifier performs worse compared to the average training dataset. Among the tested configurations, the combination of σ-VAE and the KL divergence metric outperforms the others. Therefore, it is of interest to investigate how retraining on the reconstructed samples of those identified samples affects the classifier's performance, which will be explored in the next section.

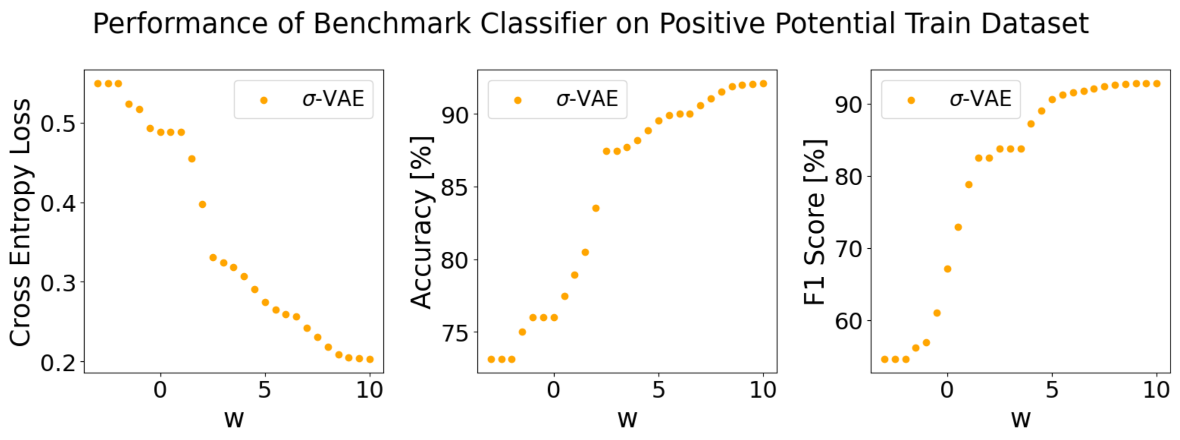

Figure 6-16: Performance results of the "20%-Classifier" on the identified positive potential dataset using σ-VAE combined with cosine similarity and KL divergence.

6-6 Evaluation of Classifier Performance with Enhanced IHDA Implementation

This section delineates the results obtained by implementing the IHDA algorithm in conjunction with σ-VAE and the VAE framework proposed by Khan and Fraz. For the σ-VAE, two different similarity metrics, namely the cosine similarity and KL divergence, were employed, while only the cosine similarity was assessed for the VAE proposed by Khan and Fraz [22]. The results from these experiments are presented in Table 6-10. The italicized entries indicate the results achieved at the optimal w value, determined through hyperparameter optimization, while the non-italicized entries represent outcomes corresponding to the w value one preceding the optimal value in each respective experiment.

Of the various tested methodologies, none of them enhanced the classifier's performance. In fact, the classifier's performance decreased compared to its performance before the application of the IHDA algorithm. For instance, the classification accuracy dropped from 85.39% to 80.51% for the combination of VAE 1 and cosine similarity, to 78.48% for the σ-VAE in conjunction with the cosine similarity metric, and further to 66.60% when the σ-VAE was combined with the KL divergence.

A consistent observation across all experiments was that the best classifier performance was achieved after just one retrain epoch, even though multiple retrain epochs were conducted. Furthermore, the optimal hyperparameter w value corresponded to the smallest positive potential dataset among the tested values of w. A significant drop in classification performance was noted when using the w value preceding the optimal one. This decline could potentially be attributed to the larger size of the positive potential dataset, leading to a larger retrain dataset. As a consequence, the IHDA algorithm implemented more parameter updates to the classifier during each retrain epoch, possibly leading to the diminished performance compared to using the optimal w value.

In summary, despite the proposed enhancements, the optimal classifier performance is achieved when the IHDA algorithm applies the fewest parameter updates to the classifier. These findings will be further examined and contextualized in the subsequent discussion.

Model	**Loss**	**Accuracy** [%]	**F1 Score** [%]	**w**	**N**$_{pp}$	Epoch
20%-Classifier	0.4667	85.39	86.42	-	-	
VAE 1 +	*0.7313*	*80.51*	*82.01*	*0.5*	*133*	*1*
Cosine Similarity	2.154	65.46	67.14	0.45	183	1
σ-VAE +	*0.6485*	*78.48*	*80.45*	*0.65*	*21*	*1*
Cosine Similarity	5.858	43.92	44.27	0.6	183	1
σ-VAE +	*1.319*	*66.60*	*67.71*	*-3*	*53*	*1*
KL divergence	4.180	40.93	36.57	-2.5	129	1

Table 6-10: Performance Metrics for Classifiers Retrained using the IHDA Algorithm with Proposed Enhancements: This table presents the validation loss, accuracy, F1 score, w values, quantity of positive potential samples (N$_{pp}$), and epochs for the original 20%-Classifier and retrained classifiers. Italicized results indicate the most favorable outcomes, identified by the lowest validation loss in each hyperparameter search, each associated with a distinct optimal w_{best} value. Non-italicized results correspond to classifiers performing best on the validation dataset when retrained with a dataset resulting from the highest tested w value preceding w_{best}.

Chapter 7

Discussion

7-1 Discussion and Interpretation of Results

This thesis aimed to implement and evaluate the proposed enhancements to the Iterative Hierarchical Data Augmentation (IHDA) algorithm, based on the original model proposed by Khan and Fraz [22]. Though it was not possible to replicate their reported results, important observations warrant discussion.

The first enhancement involved optimizing the relative weighting between the Mean Square Error (MSE) and Kullback-Leibler (KL) divergence, which was explored through the integrating σ-VAE into the IHDA algorithm. The σ-VAE demonstrated an improved reconstruction performance with reduced MSE compared to the VAE utilized by Khan and Fraz [22], which was observed during a visual comparison of samples reconstructed by the two different Variational Autoencoders (VAEs).

However, an important observation made was the mapping to zero of the accelerometer and gyroscope features for certain transportation modes, namely "Bus", "Car", "Train", "Subway", and "Still". This was attributed to their lesser intensity of movement compared to "Run", "Walk', and "Bike", affecting their contribution to the VAE loss function. This bias resulted in visually better performance in reconstructing "Run", "Walk", and "Bike" samples, while potentially compromising information about the less intense modes. Given the importance of accelerometer and gyroscope data for differentiating between transportation modes, as studied and highlighted by Friedrich et al. [22], this information loss is significant and raises questions about the efficacy of retraining on these samples.

The use of σ-VAE in conjunction with the cosine similarity metric improved the process of identifying samples for retraining. This was evidenced by a larger performance decrease of the benchmark classifier on the identified samples using the latent space of σ-VAE versus the VAE utilized by Khan and Fraz [22]. This suggests that the σ-VAE enhanced both the reconstruction performance and the interpretability of the latent space compared to the VAE proposed by Khan and Fraz [22], showcasing the efficacy of this enhancement.

The second enhancement utilized KL divergence in place of the cosine similarity metric, furthering the performance gap of the benchmark classifier on identified datasets requiring subsampling. This was interpreted as a further improvement to the IHDA algorithm's ability to identify samples requiring additional sampling.

Despite these enhancements resulting in better reconstruction performance, latent space mapping, and sample identification, they did not yield the expected performance improvement of the IHDA algorithm. On the contrary, the performance of the IHDA algorithm decreased with each proposed enhancement.
Prior research has reported increased misclassification among the motorized transportation modes [41, 42]. As each enhancement to the IHDA algorithm improved its capacity to detect misclassified samples, a corresponding rise in the frequency of samples from these motorized modes in the identified dataset could be expected. Consequently, with each enhancement of the IHDA algorithm, the identified dataset utilized for retraining could contain an enlarged proportion of samples from motorized transportation modes, which experienced a loss of distinguishing accelerometer and gyroscope features. This could provide a rational explanation for the observed decrease in performance with each adjustment to the IHDA algorithm.

Consequently, it appears that the quality of reconstructed samples introduces certain limitations that inhibit drawing definitive conclusions about the fundamental principles of the IHDA algorithm. To ascertain the suitability of VAEs for data augmentation and to investigate if the improvements reported by Khan and Fraz [22] can be achieved by comparable methodologies, which would, in turn, justify further exploration of the IHDA algorithm, a few illustrative examples from the literature are considered.

For instance, Chadebec et al. [7] proposed a VAE-based data augmentation technique where data augmentation is applied in the observational space. They validated their method's effectiveness using the MNIST image dataset, reporting a classifier accuracy increase from 78.4% to 87.6% when trained on an augmented dataset. This result provided empirical support for the attainability of the enhancements reported by Khan and Fraz [22], suggesting that an increase from 83% to 92% is plausible. However, this enhancement was observed with a training dataset containing only 20 samples per class and an augmented dataset with 12.5 times more samples. It was further observed that the effectiveness of the augmentation method declined as the number of samples per class increased. In a scenario where the training dataset contained 1000 samples per class and the augmented dataset had 12.5 times more generated samples, the performance difference between classifiers trained on the augmented and non-augmented datasets was a mere 0.4%. Nonetheless, this increase was significant, given that the classifier trained on the original dataset already achieved an accuracy of 98.5%.

This finding aligns with Kumar et al. [27], who conducted a comprehensive study on six different VAE data augmentation techniques, with some methods implementing data augmentation in the latent space. Their research confirmed that latent space data augmentation could be beneficial, depending on the method and dataset used, thereby reinforcing the core concept of the IHDA algorithm. The reported increases in classifier accuracies were for all six methods comparable to those reported by Khan and Fraz [22], although these improvements were observed for smaller training datasets with only 10 samples per class. However, Kumar et al. [27] also observed that the efficacy of the tested VAE-based data augmentation tends to diminish, resulting in minor gains as the training dataset size increases.

Both studies reported similar accuracy improvements as reported by Khan and Fraz [22] and emphasized the advantages of VAEs for data augmentation, especially when the dataset size is limited, typically comprising 20 samples per class. This observation could transform a central limitation of this study into a strength, given that it incorporated only 20 percent of the dataset used by Khan and Fraz [22].

Given reports in the literature of achieving classification accuracies above 90% with various classifiers on the utilized Sussex-Huawei Locomotion (SHL) subset [41, 40], compared to the 78% achieved in this thesis using the benchmark classifier, there is still a potential margin for improvement.

The question then arises: Are VAEs capable of introducing the necessary variation to the SHL data subset, which is relatively large compared to the examples discussed, or is an alternative network architecture necessary to achieve similar accuracies of above 90%? This forms a prospect for future investigation.

7-2 Future Work

The finding that the implemented VAE model underperformed in reconstruction within the IHDA framework, hindering a full assessment of the IHDA algorithm's effectiveness, presents opportunities for further study. The focus going forward should be on enhancing the reconstruction capabilities of the VAE model. In line with this objective, two potential areas for improvement are suggested:

- **Enhancement of the σ-VAE Model:** The σ-VAE model exhibited promising results when compared to the β-VAE model, and its elimination of the need for manual tuning is a significant advantage. To build upon these findings, further work should focus on refining the σ-VAE model. One approach worth exploring is the development of a more complex version that learns a separate decoder variance value for each feature of a sample. This would enable each feature to have an individual weight, ensuring equal significance is allocated to all features in the VAE loss function, irrespective of the MSE's magnitude. By doing so, the reconstruction of motorized transportation mode samples could be significantly improved.

- **Reassessment of Feature Processing Methodology:** Another interesting research opportunity lies in reassessing the feature processing methodology. For instance, a shift from feature-axis normalization to a sample-axis approach could be considered. This modification would involve scaling each sample's features to a range between 0 and 1 based on their individual maximum and minimum values, rather than relying on the entire feature set's range. Such an approach could better account for the unique frequency spikes associated with different transportation modes in their Fourier domain magnitudes, potentially leading to enhanced sample reconstruction quality and more effective retraining.

7-3 Conclusion

In conclusion, this thesis explored the integration of a σ-VAE into the IHDA algorithm and the replacement of the cosine similarity metric with KL divergence. While these enhancements improved the reconstruction performance of the utilized VAE and the identification of samples for retraining, they did not translate into an overall improvement in classification performance. A significant issue identified was the loss of accelerometer and gyroscope features during reconstruction, particularly for transportation modes characterized by less intense movements. Although the improvements achieved fell short of expectations, the insights gained offer important guidance for further research into refining and improving the IHDA algorithm.

Bibliography

[1] "Smartphone users 2026," Accessed:2022-04-07. [Online]. Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] "Time spent on smartphone stats 2023," Accessed:21-03-2023. [Online]. Available: https://explodingtopics.com/blog/smartphone-usage-stats#top-smartphone-stats

[3] "Track daily activity in fitness on iphone," Accessed:21-03-2023. [Online]. Available: https://support.apple.com/guide/iphone/track-daily-activity-iph9a08e004e/ios

[4] D. B. Ahmed and E. M. Diaz, "Survey of Machine Learning Methods Applied to Urban Mobility," *IEEE Access*, vol. 10, 2022.

[5] A. Antoniou, A. Storkey, and H. Edwards, "Data Augmentation Generative Adversarial Networks," *ArXiv e-prints*, Mar. 2018, arXiv:1711.04340.

[6] C. M. Bishop, *Pattern Recognition and Machine Learning.* Springer, 2006.

[7] C. Chadebec, E. Thibeau-Sutre, N. Burgos, and S. Allassonnière, "Data Augmentation in High Dimensional Low Sample Size Setting Using a Geometry-Based Variational Autoencoder," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, 2023.

[8] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," in *Proeceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

[9] G. Dorta, S. Vicente, L. Agapito, N. D. F. Campbell, and I. Simpson, "Structured Uncertainty Prediction Networks," in *Proceedings of the Conference on Computer Vision and Pattern Recognition*, 2018.

[10] I. Drosouli, A. Voulodimos, G. Miaoulis, P. Mastorocostas, and D. Ghazanfarpour, "Transportation Mode Detection Using an Optimized Long Short-Term Memory Model on Multimodal Sensor Data," *Entropy*, vol. 23, 2021.

[11] M. Frid-Adar, I. Diamant, E. Klang, M. Amitai, J. Goldberger, and H. Greenspan, “Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification,” *Neurocomputing*, vol. 321, 2018.

[12] B. Friedrich, C. Lübbe, and A. Hein, “Analyzing the Importance of Sensors for Mode of Transportation Classification,” *Sensors*, vol. 21, 2020.

[13] H. Gjoreski, M. Ciliberto, L. Wang, F. J. Ordonez Morales, S. Mekki, S. Valentin, and D. Roggen, “The University of Sussex-Huawei Locomotion and Transportation Dataset for Multimodal Analytics With Mobile Devices,” *IEEE Access*, vol. 6, 2018.

[14] ——, “The University of Sussex-Huawei Locomotion and Transportation Dataset for Multimodal Analytics With Mobile Devices,” *IEEE Access*, vol. 6, 2018.

[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Porceedings of the 27th Conference on Advances in Neural Information Processing Systems*, 2014.

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

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in *Porceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[18] S. Hemminki, P. Nurmi, and S. Tarkoma, “Accelerometer-based transportation mode detection on smartphones,” in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems - SenSys*, 2013.

[19] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, “β-VAE: Learning Basic Visual Concepts wiht a Constrained Variational Framework,” in *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[20] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” in *Proceedings of the International Conference on Machine Learning*, 2015.

[21] B. K. Iwana and S. Uchida, “An empirical survey of data augmentation for time series classification with neural networks,” *PLOS ONE*, vol. 16, 2021.

[22] A. Khan and K. Fraz, “Post-training Iterative Hierarchical Data Augmentation for Deep Networks,” in *Proceedings of the 33th Conference on Advances in Neural Information Processing Systems*, 2020.

[23] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *Proceedings of the 3rd International Conference for Learning Representations*, 2015.

[24] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” in *Porceedings of the 2nd International Conference on Learning Representations*, 2014.

[25] ——, “An Introduction to Variational Autoencoders,” *ArXiv e-prints*, 2019, arXiv:1906.02691.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Proceedings of the 25th Conference on Advances in Neural Information Processing Systems*, 2012.

[27] V. Kumar, H. Glaude, C. De Lichy, and W. Campbell, “A Closer Look At Feature Space Data Augmentation For Few-Shot Intent Classification,” in *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP*, 2019.

[28] J. Lucas, G. Tucker, R. Grosse, and M. Norouzi, “Understanding Posterior Collapse in Generative Latent Variable Models,” in *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[29] J. Lucas, G. Tucker, R. B. Grosse, and M. Norouzi, “Don’t Blame the ELBO! A Linear VAE Perspective on Posterior Collapse,” in *Porceedings of the 32th Conference on Advances in Neural Information Processing Systems*, 2019.

[30] H. Nishizaki, “Data augmentation and feature extraction using variational autoencoder for acoustic modeling,” in *Proceedings of the Asia-Pacific Signal and Information Processing Association Annual Summit and Conference*, 2017.

[31] L. E. Olsson, T. Gärling, D. Ettema, M. Friman, and S. Fujii, “Happiness and Satisfaction with Work Commute,” *Social Indicators Research*, vol. 111, 2013.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Proceedings of the Conference on Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[33] S. Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv e-prints*, Jun. 2017, arXiv:1609.04747.

[34] O. Rybkin, K. Daniilidis, and S. Levine, “Simple and effective vae training with calibrated decoders,” in *Proceedings of the 38th International Conference on Machine Learning*, 2021.

[35] J. Saldanha, S. Chakraborty, S. Patil, K. Kotecha, S. Kumar, and A. Nayyar, “Data augmentation using Variational Autoencoders for improvement of respiratory disease classification,” *PLOS ONE*, vol. 17, 2022.

[36] T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. Smith, S. Consolvo, J. Hightower, W. G. Griswold, and E. de Lara, “Mobility Detection Using Everyday GSM Traces,” in *Proceedings of the 8th International Conference on Ubiquitous Computing*, 2006.

[37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, 2014.

[38] L. Stenneth, O. Wolfson, P. S. Yu, and B. Xu, "Transportation mode detection using mobile phones and gis information," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2011.

[39] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2014.

[40] Q. Tang, K. Jahan, and M. Roth, "Deep CNN-BiLSTM Model for Transportation Mode Detection Using Smartphone Accelerometer and Magnetometer," in *Proceedings of the 6th IEEE Intelligent Vehicles Symposium)*, 2022.

[41] L. Wang, M. Ciliberto, H. Gjoreski, P. Lago, K. Murao, T. Okita, and D. Roggen, "Summary of the Sussex-Huawei Locomotion-Transportation Recognition Challenge," in *Proceedings of the ACM International Joint Conference and International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, 2018.

[42] L. Wang, H. Gjoreski, M. Ciliberto, S. Mekki, S. Valentin, and D. Roggen, "Benchmarking the SHL Recognition Challenge with Classical and Deep-Learning Pipelines," in *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, 2018.

[43] L. Wang, H. Gjoreski, M. Ciliberto, P. Lago, K. Murao, T. Okita, and D. Roggen, "Summary of the Sussex-Huawei locomotion-transportation recognition challenge 2019," in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the ACM International Symposium on Wearable Computers*, 2019.

[44] L. Wang, H. Gjoreski, K. Murao, T. Okita, and D. Roggen, "Summary of the sussex-huawei locomotion-transportation recognition challenge 2020," in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the ACM International Symposium on Wearable Computers*, 2020.

[45] L. Wang, M. Ciliberto, H. Gjoreski, P. Lago, K. Murao, T. Okita, and D. Roggen, "Locomotion and Transportation Mode Recognition from GPS and Radio Signals: Summary of SHL Challenge 2021," in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the ACM International Symposium on Wearable Computers*, 2021.

[46] S.-C. Wang, *Interdisciplinary Computing in Java Programming.* Kluwer Academic Publisher, 2003.

[47] Q. Wen, L. Sun, F. Yang, X. Song, J. Gao, X. Wang, and H. Xu, "Time Series Data Augmentation for Deep Learning: A Survey," in *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, 2021.

[48] P. Widhalm, P. Nitsche, and N. Brändie, "Transport mode detection with realistic Smartphone sensor data," in *Proceedings of the 21st International Conference on Pattern Recognition*, 2012.

[49] Y. Xiao, D. Low, T. Bandara, P. Pathak, H. B. Lim, D. Goyal, J. Santos, C. Cottrill, F. Pereira, C. Zegras, and M. Ben-Akiva, "Transportation activity analysis using smartphones," in *Proceedings of the IEEE Consumer Communications and Networking Conference*, 2012.

[50] Y. Zheng, L. Liu, L. Wang, and X. Xie, "Learning transportation mode from raw gps data for geographic applications on the web," in *Proceedings of the 17th International Conference on World Wide Web*, 2008.

Glossary

List of Acronyms

TMD	transportation mode detection
SHL	Sussex-Huawei Locomotion
ANN	Artificial Neural Network
ML	Machine Learning
DL	Deep Learning
DA	Data Augmentation
GAN	Generative Adversarial Network
VAE	Variational Autoencoder
AE	Autoencoder
PCA	Principal Component Analysis
MSE	Mean Square Error
KL	Kullback-Leibler
ELBO	Evidence Lower Bound
ReLU	Rectified Linear Unit
DNN	Dense Neural Network
SGD	Stochastic Gradient Descent
GD	Gradient Descent
IHDA	Iterative Hierarchical Data Augmentation
RBF	radial basis function
IHDA	Iterative Hierarchical Data Augmentation
NeurIPS	Conference on Neural Information Processing Systems