UNIVERSITY
OF OSLO

Master’s thesis

Self-supervised feature extraction from video and vehicle monitoring sensors for autonomous driving

Andreas Sykas Sundfjord

Robotics and Intelligent Systems
30 ECTS study points

Departement of Informatics
Faculty of Mathematics and Natural Sciences

Spring 2023

Andreas Sykas Sundfjord

Self-supervised feature extraction from video and vehicle monitoring sensors for autonomous driving

Abstract

The development of autonomous vehicles and Advanced Driver Assistance Systems (ADAS) relies heavily on the accurate interpretation of driving scenes. This task requires identifying and tracking multiple objects such as traffic signs, pedestrians, and other vehicles. Moreover, it is essential for these systems to accurately recognise various driving scenarios, such as entering intersections and yielding to pedestrians. Current state-of-the-art applications utilise deep learning methods and often need vast amounts of labelled data, which is expensive to acquire. A possible solution to this problem is applying self-supervised learning methods to extract features from driving scenes. This approach does not need labelled data and has high scalability. The extracted features possess the potential for utility in various other applications, such as recognising driving scenarios.

The main goal of this thesis is to extract features from unlabelled data and assess whether the extracted features can be generalised for a broad range of applications. This data consists of real-world driving scenes, which are recorded with a front-facing camera and vehicle monitoring sensors. To extract the features, various implementations of Autoencoders are utilised and compared. Two primary tasks are selected to evaluate the extracted features: clustering and classification. Clustering is used to identify similar driving scenarios. The classification task is implemented for types of driving scenarios and perceived risk for driving scenes. In addition, the work evaluates whether incorporating input from multiple sensors can improve performance in these tasks.

The work found that combining the video and vehicle monitoring sensor data using mid-level sensor fusion achieved a silhouette score of 0.34 for K-Means clustering with four clusters, thus enhancing the performance of single modality implementations. A different implementation utilising attention layers improves performance in a 3-class classification task, achieving an accuracy of 38.2%.

Acknowledgements

This thesis was written over one semester as part of an international collaboration with the Behavior Signal Processing Laboratory of Nagoya University in Nagoya, Japan through the Collaboration on Intelligent Machines 2 (COINMAC-2) project. I would like to thank my supervisor Jim Tørresen for arranging my stay at this laboratory and supporting me with guidance and feedback during my stay.

I would also like to thank the Behavior Signal Processing Laboratory and Kazuya Takeda for allowing me to use the facilities and resources of the laboratory and making sure I settled in. I am grateful for the help and guidance of Kento Ohtani and Alexander Carballo, whose experience and assistance proved invaluable.

Further, I would also like to thank PhD students Robin Karlsson and Kyle Sama for welcoming me and assisting my work with their experience and ideas.

This work is partially supported by The Research Council of Norway (RCN) as a part of a grant to undertake the master project abroad through the (COINMAC-2) project, under grant agreement no. 309869. The work is also supported by National Research and Development Agency-New Energy and Industrial Technology Development Organization (NEDO) in Japan.

Finally, I would like to thank my family and loved ones for their support throughout my studies. Without them, none of this would have been possible.

Contents

List of Figures

List of Tables

List of Algorithms

List of Equations

Acronyms

ADAS Advanced Driver Assistance Systems

AE Autoencoder

AI Artificial Intelligence

ANN Artificial Neural Network

ARI Adjusted Rand Index

CAN Controller Area Network

CE Cross-Entropy

CH Calinski-Harabasz

CNN Convolutional Neural Network

CPC Contrastive Predictive Coding

GANs Generative Adversarial Networks

GMM Gaussian Mixture Model

GPS Global Positioning System

GRU Gated Recurrent Unit

HVAE Hierarchal Variational Autoencoder

KL Kullback–Leibler

KNN K Nearest Neighbors

LiDAR Light Detection and Ranging

LR Learning Rate

LSTM Long Short-Term Memory

MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MSE	Mean Square Error
MSVAE	Multi-Modal Semi-Variational Autoencoder
MTVAE	Multi-Modal Transformer Variational Autoencoder
MVAE	Multi-Modal Variational Autoencoder
NEDO	National Research and Development Agency-New Energy and Industrial Technology Development Organization
NLP	Natural Language Processing
PCA	Principal Component Analysis
RADAR	Radio Detection And Ranging
RI	Rand Index
RNN	Recurrent Neural Network
SSL	Self-Supervised Learning
SVM	Support Vector Machine
TTC	Time-To-Collision
TVAE	Transformer Variational Autoencoder
VAE	Variational Autoencoder
VMS	Vehicle Monitoring Sensor

Chapter 1

Introduction

This chapter introduces the outline and goals of this thesis. **Section 1.1** provides some background on autonomous driving and describes what the thesis aims to achieve. **Section 1.2** states the thesis goals and research questions. **Section 1.3** expresses the scope and limitations of the work. **Section 1.4** explains the benefits of achieving the goals, and states the contributions of this thesis. Finally, **section 1.5** outlines the structure of the thesis.

1.1 Motivation

An important area of focus in the Artificial Intelligence (AI) and Machine Learning (ML) sectors is autonomous driving. This refers to vehicles that can operate to a certain degree without the need for human intervention. A fully self-driving vehicle must be able to navigate any kind of road without human assistance or intervention. The emergence of this industry has the potential to provide significant benefits to both the private and public sectors [1, 2, 3]. One of these benefits is reducing motor accidents, where Treat [4] estimated that human error was a definite cause in 70% of accidents. Treat et al. [5] also estimated that human factors were a probable cause in 92.6% of motor accidents. Causes include falling asleep while driving [6], texting and driving [7], and driving under the influence of alcohol. Hence, the implementation of autonomous vehicles could drastically reduce the number of motor accidents in the world. Autonomous driving also offers the advantage of providing greater independence to individuals who are unable to drive themselves, including elderly and disabled persons, in addition to enhancing transportation accessibility overall [8]. Other potential benefits included reduced energy consumption and pollution [8]. However, as further explained in section 2.1, technological advancements must be made before fully self-driving cars are ready to be made available.

This thesis attempts to extract features from driving scenes using Self-

Supervised Learning (SSL), which is a process where an ML model is trained on *unlabelled* data. Specifically, the chosen approach utilises an Autoencoder (AE) model to encode input data to a simpler, lower-dimensional representation that contains the extracted features. This method of feature extraction is described in more detail in section 2.4, and the implementation is explained in section 4.2. Being able to obtain features using only unlabelled data is highly beneficial, due to the time-consuming process of annotating data.

The input data is acquired from a front-facing *camera* mounted on the vehicle, and various *Vehicle Monitoring Sensors* (VMS). This is referred to as modalities, which means how something is experienced or expressed. The primary focus of this thesis is to compare and evaluate different approaches to encoding and combining these modalities. The camera image provides detailed descriptions of the scene in front of the vehicle, while the VMS data provides information about the recording vehicle, driver, and surroundings. In addition, the VMS data includes detected vehicles, pedestrians, and bicycles in the driving scene, which means that this work can focus on applying that information to other tasks, instead of training a model for object detection.

The work aims to show that the representations can be used to group similar driving scenes by comparing the extracted features, which demonstrates that the model can distinctly encode and represent features from different scenes. This process is known as *clustering* and is explained in subsection 2.5.1. However, a challenge with this evaluation approach is ensuring the quality of the clusters, and understanding what these clusters represent in terms of real-world driving scenes. This challenge is addressed in subsection 2.5.1. Clustering is also useful for anomaly detection, in addition to easily assigning a newly recorded driving scene to a group of similar ones. This thesis also explores the viability of using the extracted features from an AE model as input for a supervised classification model to assign labels. The predicted labels include the classes that are explained in Table 3.1 and the risk level of the driving scenes. This is a process known as *fine-tuning*, which often performs better when data is limited compared to supervised ML methods [9]. Tasks like clustering and fine-tuning are often referred to as downstream tasks. This is a general term for tasks that involve using the output of one task as the input for another, such as training a model for reconstructing data and using the encoded data for a clustering task. This essentially means applying previously acquired knowledge to solve a new problem.

To summarise, the thesis explores using different self-supervised AE models for extracting features from video and VMS data. Further, the work determines which of the tested implementations performs the best for two tasks, clustering and classification. This is tested using the NEDO dataset, which is a collection of real-world driving scenes in city traffic. The dataset contains annotations for types of driving scenarios, perceived risk, action tags, environment tags and captions.

1.2 Research Questions

The goal of this work is to use sensor data from both a colour camera and vehicle monitoring sensors and extract features from this data using Self-Supervised Learning. The aim is to represent the extracted features from driving scenes as a vector and ensure that it is generalised, to be used for both clustering and classification tasks. With this goal in mind, this thesis seeks to answer the research questions given below:

RQ1 How can features be extracted from driving scene sensor data using self-supervised methods?

RQ2 What is an effective method to combine video and Vehicle Monitoring Sensor data?

RQ3 Will using both modalities improve performance in clustering and classification of extracted features?

This thesis aims to answer these questions by evaluating different variations of ML models. The evaluation will examine how well the produced representations generalise for different tasks, and compare the performance of the implemented models. The tasks include grouping driving scenes by similarity and predicting both the type of driving scenario and the perceived risk.

1.3 Scope and Limitation

Amongst the limitations of this thesis are both the amount of variations of implementations, and the different applications of the extracted features from the driving scenes. Improvements could be made to the models, data pre-processing techniques, and evaluation methods. However, the main goal of this thesis is to show that a latent representation of multi-modal data can be used for clustering and classification tasks and to compare different ways of modality encoding and combination. Hence, optimising performance is not a requirement and the goal is not to reach state-of-the-art performance. However, it is essential to attain a level where the latent representations of an individual are informative enough to have a positive impact. By achieving this, it becomes possible to compare the different approaches. This work is therefore limited to testing the viability of these methods with limited data.

The annotations in the utilised dataset present many possible applications for downstream tasks, such as action prediction, environment tag generation, and captioning. However, this work will only use the annotation for types of driving scenarios and risk scores for classification tasks.

1.4 Contributions

The main contributions of this thesis are:

- A comparison of different implementations of encoding and combining video and data from vehicle monitoring sensors.
- Further research on the viability of self-supervised methods for extracting features from the two modalities. Specifically, the use of Autoencoders for this purpose.
- Evaluate the accuracy of extracted information from a model trained with limited data.

Although the research questions are evaluated using driving scenes, the proposed methods can in theory be applied to other areas of robotics that utilise similar sensors.

1.5 Thesis Structure

The structure of this thesis is as follows:

- **Chapter 1** provides background information about the field of autonomous driving and introduces what the thesis aims to do.
- **Chapter 2** expands on previous related research in this field, and the methods on which the experiments build. This includes sensors used in autonomous driving, common ML methods for SSL and evaluation of results.
- **Chapter 3** explains the dataset that is used in this thesis, what it includes and its limitations, and how the data is pre-processed.
- **Chapter 4** expands on the methods used in the experiments for this thesis, including how the ML models and the evaluation techniques are evaluated.
- **Chapter 5** presents the experiments conducted in this report, including why they are done, what the results are, and an analysis of these results.
- **Chapter 6** describes potential future work, regarding method improvements and additional applications.
- **Chapter 7** contains the conclusion of the thesis.

Chapter 2

Background

This chapter reviews the relevant literature for this thesis, and the theoretical framework the work is based on is explained. **Section 2.1** briefly reports the current state of the autonomous driving research field. **Section 2.2** describes sensors commonly used in autonomous driving and how they have been utilised together. **Section 2.3** describes common ML models, and **section 2.4** concerns self-supervised feature extraction methods. Lastly, **section 2.5** explores methods to evaluate extracted features.

2.1 Current State of Autonomous Driving

This section briefly describes the current state of autonomous driving, in addition to providing background on the technologies involved. Autonomous driving builds on two key techniques: real-time communication between vehicles [10] and the system's ability to observe and comprehend traffic scenes [11]. The latter is the focus of this thesis. It is a task that has been approached by applying ML methods, often utilising supervised methods that need annotated data [12, 13]. To function effectively, an autonomous driving system must be able to identify important objects, including vehicles, pedestrians, and traffic signals as shown in Figure 2.1. This is an important research topic in autonomous driving where accuracy is critical [14, 15, 16]. Building on this, there are different approaches to training ML models to understand driving scenes. One could classify driver actions and the reasoning for performing them [12]. Another alternative is to classify potential dangers, where they are, and how important they are [17]. Thus, when a system is capable of understanding driving scenes, it can make informed decisions and operate autonomously.

There are several definitions of autonomous driving, often expressing different extents of autonomy. A common description, proposed by the Society of Automotive Engineers, divides technologies into different levels of automated driving [18]:

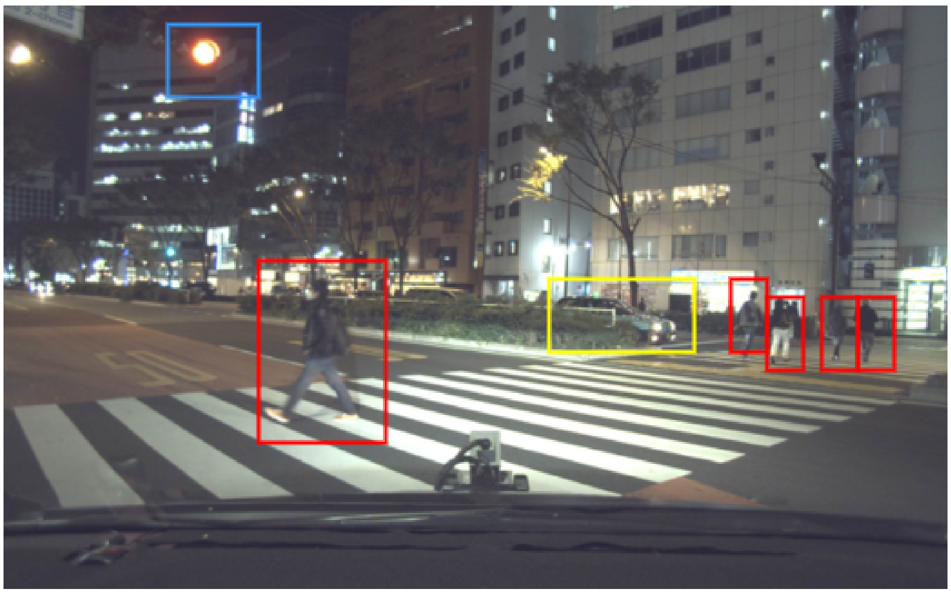


Figure 2.1: Detected objects in a driving scene. The objects that are relevant to an autonomous driving system are marked by bounding boxes.

1. **No automation**. At this level, there is no automation, and the driver performs every action unassisted.
2. **Driving assistance**. There is some assistance in steering, braking, or accelerating. An example of this is adaptive cruise control, which automatically keeps a safe distance from the vehicle in front.
3. **Partial Driving Automation**. This level applies to vehicles with Advanced Driver Assistance Systems (ADAS)[1], with systems that can take over steering, acceleration, and braking. The driver is still required to actively supervise the vehicle at this level.
4. **Conditional driving automation**. At this level, the driver does not need to supervise the system. However, the driver is required to stay alert and be ready to take control of the vehicle at all times. This level is constrained to specific situations and locations.
5. **High driving automation**. At this level, the driver is not needed at all and is not required to take control of the vehicle at any point. This level is limited to specific locations and conditions.
6. **Full driving automation**. At the final level, the vehicle can operate automatically at any location, under any condition.

[1]ADAS is meant to assist human drivers and prevent accidents.

Although level 3 systems exist and have been certified in some countries [19], they are not mass-produced nor commonly found as of June 2023. Significant technological advancements must be made before fully-self driving vehicles at levels 4 and 5 are safe. Even then, there are significant challenges that need to be addressed regarding security and privacy [20], in addition to concerns about ethics, policies, strategies, and liability [21].

2.2 Sensors Used in Autonomous Driving

As mentioned in section 1.1, understanding and observing the surroundings is crucial for any autonomous driving technology. To achieve this, various sensors are utilised, each with its specific purpose. This section discusses commonly used sensors and their functions. Additionally, an essential aspect of autonomous driving research is finding ways to effectively integrate and use these sensors to enhance decision-making, predict potential risks, and gain a better understanding of driving scenarios.

Colour Camera One of the most common sensors is the colour camera. They are widely used due to their cost-effectiveness and ability to provide high-quality information. Often, vehicles are mounted with multiple cameras facing in different directions to give gain a comprehensive understanding of the driving scene and detect objects 360 degrees around the ego vehicle[2]. However, there is an additional computational cost for the challenging task of assembling the images from various cameras. Despite their usefulness, colour cameras have limitations, including the inability to provide information about an object's distance and susceptibility to weather conditions like varying illumination, rain, and snow.

Depth Sensors There are many kinds of depth sensors available, such as Light Detection and Ranging (LiDAR), Radio Detection And Ranging (RADAR) and stereo cameras. These sensors provide information about the distance from objects to the sensor. Some depth sensors are also more robust to the effects of illumination and weather conditions than colour cameras. As regular colour cameras do not provide information about the distance to objects, these two sensors are often combined [22, 23].

Thermal Camera A thermal camera captures light at a near-infrared range and measures the temperature that is emitted from objects. Hence, thermal cameras use the heat emitted from vehicles and humans to detect them. This is especially useful in low-illumination scenarios, such as driving scenes during night time, and situations where vision is occluded by dust

[2]Ego vehicle refers to the vehicle that is recording the driving scene.

and fog. This covers an area where colour cameras struggle to capture useful information, sparking an interest in researching the combination of the two sensors [24].

Vehicle Sensors Information can be collected by sensors both inside and outside of the vehicle. The Controller Area Network (CAN) protocol is commonly used to monitor the performance and safety of a vehicle by providing data on velocity, acceleration, and controls. Additionally, GPS can be utilised to gather similar information and track the position of the vehicle. An example of successful implementation of vehicle sensor data comes from research by Zhang et al. [13], who used sensor information in conjunction with video from a front-facing camera to improve performance in a video captioning task. By supplementing the video with vehicle sensor data, they were able to obtain information about the driver's actions even though the vehicle itself was not visible in the footage.

AI-Based Sensors Modern vehicles often have systems on board that are based on AI models and algorithms. The systems process raw images and output object detection results, object trajectories, Time-To-Collision (TTC), and other useful information. An example of an AI-based sensor is Mobileye, which is often used in ADAS. However, it is important to ensure the accuracy of the model outputs. It has been shown that the Mobileye sensors can be fooled by projecting traffic signs on other vehicles [25]. However, for normal driving scenarios, Mobileye and other AI-based sensors can be deemed trustworthy.

2.2.1 Sensor Fusion

This part explores the different ways sensor fusion has been researched in the ML field. It is an important area of research, both regarding autonomous vehicles, and other topics [26, 27]. In their review, Yeong et al. [28] describe and evaluate different approaches for multi-sensor fusion in driving scenes. Combining input from different sensors often requires initial input processing, as modalities may have varying representations. Yeong et al. [28] categorise sensor fusion into three approaches: low-level, mid-level, and high-level fusion. Low-level fusion involves combining the raw data with minimal pre-processing, high-level fusion involves combining the extracted features from each sensor, and mid-level is somewhere in between. While high-level fusion has the advantage of being the least complex method, Yeong et al. [28] state that low-level fusion often achieves better performance. Tellamekala et al. [29] propose a novel sensor fusion framework, which combines different encoded modalities at a high level while taking into account the uncertainty of each modality, showing that good results can be achieved with high-level modality fusion.

Numerous researchers have delved into the realm of multi-modal sensor fusion, with several studies regarding autonomous driving [30, 28]. As shown in the review by Rizzoli, Barbato and Zanuttigh [31], multiple papers have researched combining colour cameras and other sensors to improve the performance of semantic segmentation[3]. A popular sensor combination involves a colour camera and a depth sensor, which work together to provide information on an object's appearance and location [32, 33]. Researchers have also successfully paired colour cameras with thermal cameras to enhance performance in detecting objects under low-illumination conditions. This has been achieved through the implementation of various mid-level fusion techniques, including graph attention networks [34] and summation [35]. It has also been found that combining sensor data from a colour camera with VMS data improves accuracy in a classification problem for understanding real-world driving scenes [12]. Seeking to understand driving scenes, Ramanishka et al. [12] propose a novel 4-level representation to describe driver decisions:

- **Goal-oriented.** An action made to reach a goal. This could mean turning left to reach the destination.
- **Stimulus-driven.** An action made responding to stimuli. This could mean stopping as a pedestrian crosses the road.
- **Cause.** The cause of the stimulus-driven action. This could be traffic congestion or a crossing pedestrian. This is marked by a bounding box.
- **Attention.** When driving, humans pay attention to other traffic participants. This level of the 4-level representation uses a bounding box to mark other traffic participants, such as a pedestrian walking on the sidewalk.

[12] combine the two modalities at mid-level, by concatenating the feature vectors. In their paper, Huang et al. [36] propose a novel algorithm for combining multi-modal sensor input for end-to-end driving and scene understanding. They used simulated driving scene data gathered using CARLA[4], and found a performance improvement when using a multi-modal sensor fusion encoder network. Huang et al. [36] used colour images, with a depth image layered on top, as input for the encoder network. The resulting combined encoded representation was used for modelling a driving policy. Thus, it is well documented that multiple modalities can increase performance in autonomous driving tasks, as opposed to relying on input from a single type of sensor.

[3]Semantic segmentation is the process of assigning a label to every pixel in an image or a video

[4]CARLA is an open-source simulator for autonomous driving research and stands for Car Learning to Act.

2.3 Artificial Neural Networks

This section explains the different types of ML models relevant to this thesis, and why they are used. The different models can be used alone, or together to form more complex models.

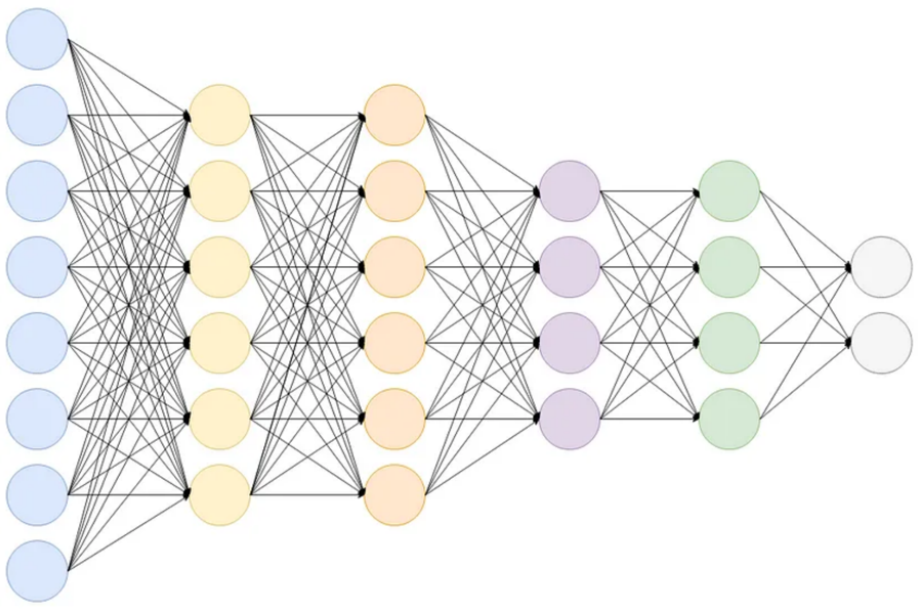


Figure 2.2: Artificial Neural Network containing fully connected layers [37]. The blue nodes represent the input data, which is passed through hidden layers before the output is eventually presented in the final grey layer. The connections between the nodes represent weights.

The Artificial Neural Network (ANN) is the basis of deep learning algorithms in ML and is loosely inspired by the structure and function of biological neurons in the brain. In an ANN, the model consists of multiple layers of artificial neurons, often called nodes, which are connected via a set of weights. Each neuron receives input from the previous layer, applies a non-linear activation function to the input, and passes the output to the next layer. A common layer is a fully connected layer, as illustrated in Figure 2.2.

Training a model When training an ML model, the training objective is formulated as a loss function. Loss functions often compare the difference between predicted and actual values to optimise a training objective, from which gradients are calculated. The gradients are then used to update the model weights in a way that decreases the loss, thus decreasing said difference. Examples of loss functions are Mean Square Error (MSE) and Cross-Entropy (CE). The formula for MSE is shown in equation 2.1, which calculates the square difference between the model output and true value. It is one of the most common loss functions used in ML and is used to minimise this difference.

$$MSE = \sum_{i=1}^{D}(x_i - y_i)^2 \tag{2.1}$$

For classification tasks, CE loss is often used. The variant of cross entropy loss shown in equation 2.2 is for when there are more than 2 classes. Essentially it returns the natural logarithm of the probability of the correct class. When predicting classes, there is not necessarily an inherent similarity between classes, as there is with numeric values. Therefore, CE loss is used to maximise the probability of the true class.

$$CE = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{2.2}$$

L2 regularisation, also known as Ridge regression, encourages smaller weights in the model. As shown in equation 2.3, the squared value of all the weights in the model is summed, and multiplied by a factor lambda. This is used in addition to another loss function and helps prevent overfitting the model to the training data.

$$L2 = \lambda \sum_{1}^{n} w_i^2 \tag{2.3}$$

2.3.1 Spatial Data Models

One type of neural network that is commonly used is called a Convolutional Neural Network (CNN). This network is specifically designed to identify spatial patterns within data. The CNN typically consists of convolutional layers that extract features from the data, followed by pooling layers that reduce the size of the feature maps. In Figure 2.3, you can see how a pixel in one layer takes a group of nearby pixels from the previous layer as input. A convolutional operation is then performed to represent these pixels as a single value. This allows for context to be taken from nearby pixels, which makes it a great tool for processing images. The CNN can also be used to process and extract features from videos by using 3D convolution.

2.3.2 Sequential Data Models

The Recurrent Neural Network (RNN) is a type of neural network specifically designed to process sequential data, like human language, making them

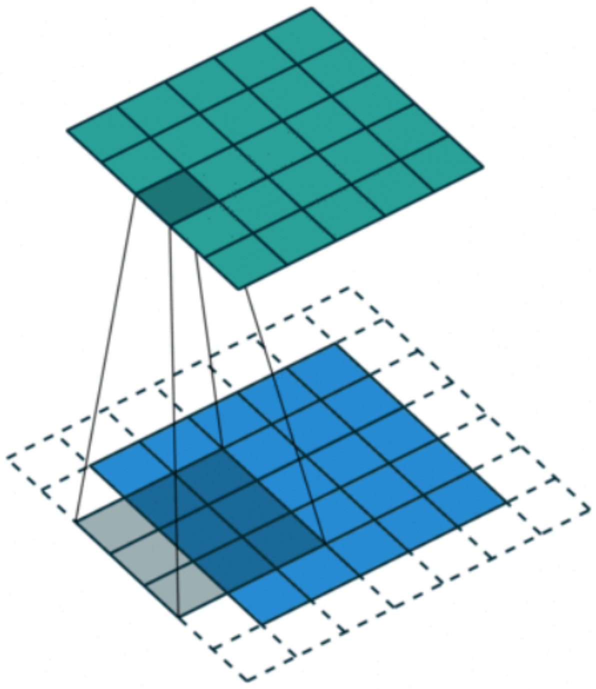


Figure 2.3: Convolutional Neural Network layer [38]

a popular choice in Natural Language Processing (NLP). RNNs are designed to recognise temporal patterns in the input data by passing information from one time step to the next. The network is made up of a series of repeating units, each processing a time step of the input sequence, as shown in Figure 2.4. The figure illustrates how the RNN takes input at each timestep, processes it, and passes it to both the output layer and the next timestep. This allows information from previous timesteps to influence future ones.

An issue that arises with RNNs is the vanishing gradients problem, where the gradients become too small to significantly update the model weights. This can occur when the gradients are consecutively passed through the repeating units and effectively stops the model from training further. The GRU model addresses this issue by controlling which information is passed on to the output [40]. Other benefits of the GRU models are faster training time and smaller memory usage. Another RNN variation is the Long Short-Term Memory (LSTM) model, proposed by Hochreiter and Schmidhuber [41]. It has been shown to outperform the GRU model in certain scenarios, depending on sequence length and dataset size [42].

Transformers Although LSTMs are more accurate with longer sequences than regular RNN and GRU models, they still have limitations. A Trans-

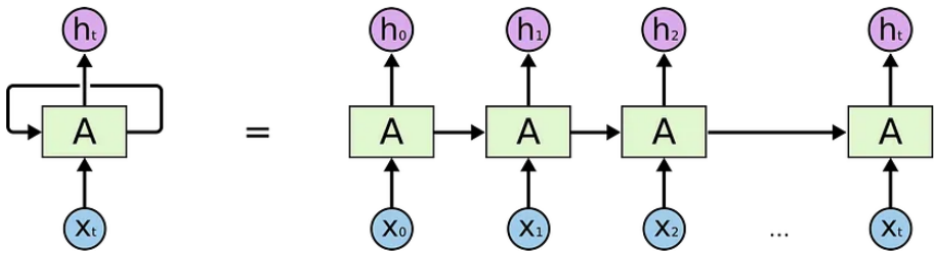


Figure 2.4: Recurrent Neural Network repeating units [39]

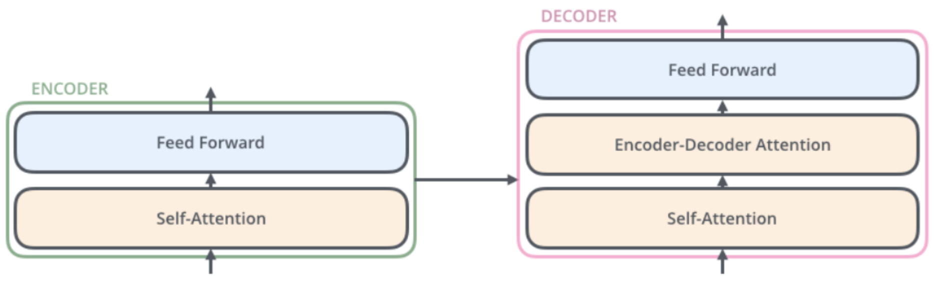


Figure 2.5: Components in transformer encoder and decoder [48]

former is another type of neural network architecture that was proposed in 2017 by Vaswani et al. [43], which proved to outperform RNNs in various tasks [44]. Transformers are primarily used for NLP tasks such as language translation, language modelling, and text classification, but have been found to perform well with video data in various tasks as well [45, 46]. They have been used in SSL for combining sequential multi-modal data, showing promising results [47].

The key innovation of the Transformer is the self-attention mechanism, which allows the model to capture long-range dependencies between input tokens without the need for recurrent layers. Self-attention works by computing a weighted sum of the input tokens, where the weights are learned based on the similarity between the tokens. The use of an attention mechanism is widespread in many neural networks. It enables the model to focus on and remember parts of the input that are deemed important, ultimately improving accuracy and robustness on various tasks. Due to its effectiveness, the attention mechanism has become a popular choice for numerous ML applications.

Transformers employ stacked encoders and decoders with identical structures but without shared weights. These structures are depicted in Figure 2.5. Each encoder feeds the output to the next one until the last encoder passes the output to the decoders.

2.4 Self-Supervised Feature Extraction

This section presents algorithms and models proposed to extract feature information. Feature extraction involves converting raw data into representative numerical values while retaining the information from the original data. This work researches extracting feature information without using the labels of the data set, thus focusing on self-supervised methods, as stated in RQ1 in section 1.2. The basis of the proposed models in this thesis is the AE, which is explored further in the next subsection. There are many AE variations, such as convolutional AEs [49], graph AEs [50], and recurrent AEs [51]. The convolutional and recurrent autoencoders are especially relevant to this thesis, as the modalities used are both video and sequential numerical data. Other methods include Generative Adversarial Networks (GANs), which are often used for data generation, and Contrastive Predictive Coding (CPC) [52], which predicts past or future data points in sequential data. AEs can be trained as a foundation for downstream tasks [53], which makes them a promising method regarding RQ1.

Latent space representations Extracted features from raw data can be referred to as latent representations. A latent representation is a fundamental concept in ML that refers to a set of hidden or unobserved variables that capture the underlying structure and patterns in the data. The goal is to have similar data samples positioned closely in the latent space, thus having similar latent representations. This concept is often used in the context of SSL, where, as mentioned in section 1.1, the objective is to learn a compact, lower-dimensional representation of the input data.

2.4.1 Autoencoders

As stated in section 1.1, this thesis will look at how AEs can be used to understand driving scenes, specifically how they can be used to obtain latent representations of multi-modal data. The AE is a type of neural network that is designed to learn a compressed representation of input data and is often used in SSL tasks. The architecture of an AE typically consists of an encoder, which maps the input data into a lower-dimensional latent space, and a decoder, which reconstructs the original input from the latent space representation. By minimising the difference between the input and the reconstructed output, the AE learns to capture the most important features of the data in the latent space. These features can then be used for tasks such as classification [55], clustering [56], and prediction [57].

Figure 2.6 illustrates a general AE structure. The layers can be of any type, such as the aforementioned CNN, RNN, and fully connected layers. Note that the encoded vector referenced as "Code" in the illustration, is referenced as the latent representation in this thesis. The following paragraphs

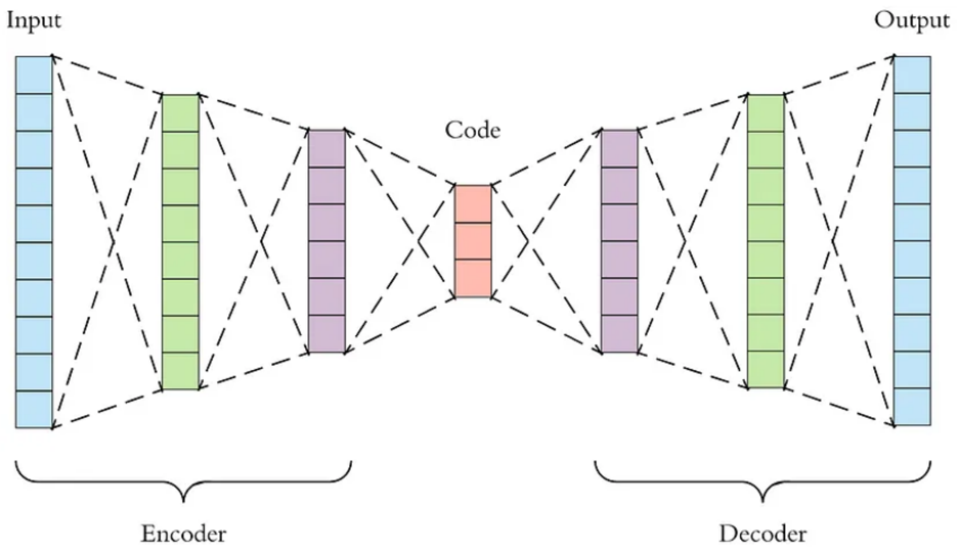


Figure 2.6: General autoencoder architecture [54]. The encoder and decoder are built by neural network layers, which can be fully connected, convolutional, or recurrent layers. The encoded representation in the middle has fewer dimensions than the input data.

describe relevant AE variations.

Variational Autoencoder One of these variations is the Variational Autoencoder (VAE), which regularises the encoded vectors to ensure that the latent space contains good properties. This can be achieved by utilising Kullback–Leibler (KL) divergence, which is used to minimise the difference between the distribution of the encoder output and a given distribution. Adding this to a loss function will help to regularise the latent space. The formula used to calculate the KL divergence is shown in equation 2.4, where $\hat{y}$ is the latent space distribution, and y is the desired distribution to approximate. When approximating a standard multivariate normal distribution, the formula can be calculated as shown in equation 2.5, where μ is the mean and σ^2 is the variance of the latent distribution. This can be achieved by training two separate fully connected layers, one for the mean and one for the variance, instead of just one fully connected layer for the encoded output.

$$KL(\hat{y}||y) = \sum_{c=1}^{M} \hat{y}_c \log \frac{\hat{y}_c}{y_c} \tag{2.4}$$

$$KL = -0.5 * \sum_{c=1}^{M} 1 + log(\sigma^2) - \mu^2 - e^{log(\sigma^2)} \tag{2.5}$$

Using KL divergence in the loss computation prevents overfitting the latent space, ensuring that a latent representation does not only contain the information necessary to reconstruct the original data sample. Because of this property, VAEs have been used to generate new data samples, and shown to meet state-of-the-art data generation methods [58].

Another benefit of the property of the latent space produced by VAEs is the ability to handle incomplete input data. In many cases, the input data can lack sequences of data or even features altogether. In their paper, Yu et al. [59] show that using a Variational Autoencoder (VAE) for massive data wireless multimedia sensor networks can help in extracting features from incomplete data.

Hierarchical VAE Another type of AE is the Hierarchal Variational Autoencoder (HVAE), which is an extension of the VAE model that adds a hierarchical structure to the latent space. In a HVAE, the latent space is divided into multiple levels, with each level representing a different level of abstraction or complexity. The lower levels capture more detailed local features of the data, while the higher levels capture more global features. This allows the HVAE to capture more complex structures of information in the latent representations. However, they also require more training time and computational resources due to their hierarchical structure. Karlsson et al. [60] propose a method to predict complete states from incomplete input for real-world modelling, using a HVAE.

Multi-modal Autoencoder The final AE variation to be discussed is the Multi-Modal Variational Autoencoder (MVAE). Essentially, a MVAE combines input data from multiple sensors. The advantage of this method is the added context from other modalities, both in reconstructing original samples, and forming a more descriptive latent space. As described in subsection 2.2.1, the combination of modalities can be done at low-, mid- and high-level. The MVAE can also be combined with the previously discussed variations, such as VAEs and HVAEs [61, 62].

2.5 Evaluation of Extracted Features

In this section, both supervised and self-supervised approaches to evaluating latent space representations are presented.

2.5.1 Clustering

Clustering is an unsupervised method, which groups data samples based on similarity. There are different methods to compute and determine this similarity.

K-Means A common clustering method is the K-Means algorithm. This algorithm starts by randomly selecting K cluster centres, and then iteratively assigns each latent representation to the nearest centre. The cluster centres are then updated to be the mean of the samples in the cluster. The process continues until the centres no longer move, the change is below a threshold, or a maximum number of iterations is reached. It is a simple, yet effective algorithm for measuring how well the latent representations are separated. However, the decision boundaries between cluster centres are linear, which might not adequately separate the samples. If this is the case, other clustering algorithms should be considered.

Spectral clustering Spectral clustering stems from graph theory and assigns clusters to nodes based on the edges connecting them. This technique works well for non-linear data structures and handles high-dimensional data well. In short, the method consists of calculating a weighted adjacency matrix of the data points, using this to compute the graph Laplacian matrix for the data. Then, the eigenvalues of the graph Laplacian matrix are calculated and used to embed the data points to a lower dimensional space. The resulting embedded data can then be used for a simpler algorithm, such as K-means. Spectral clustering has been found to produce more distinct clusters than the K-Means algorithm in previous research [63].

Gaussian Mixture Models There are some limitations to the K-Means algorithm, one of them being that it is a hard clustering algorithm, meaning it assigns each data point to only one cluster. Although the Gaussian Mixture Model (GMM) method is similar, it bases the clusters on probability instead of distance. The method assumes that the data samples are from multiple Gaussian distributions, which can be interpreted as clusters. The illustrations in Figures 2.7 and 2.8 visualise how two distributions can be used to form clusters. Patel and Kushwaha [64] found that using GMM for clustering provided better results and more distinct boundaries than the K-Means algorithm. GMMs have also been used for classification tasks [65].

Cluster Evaluation For clustering tasks, a metric should measure the compactness of the clusters, and the separation of different clusters. If labels are available, metrics that measure how the predicted labels match the true labels can be used. Some of the common metrics are described below.

One option for self-supervised metrics is the silhouette score, which calculates the mean within-cluster distance a and the mean nearest-cluster distance b. The score is then computed as seen in Equation 2.5.1, taking the mean score of all samples. The best possible score is 1, and the worst is -1. Values near 0 indicate overlapping clusters and negative values generally

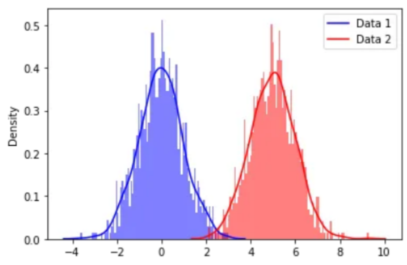


Figure 2.7: Mixture of two Gaussian distributions [66]

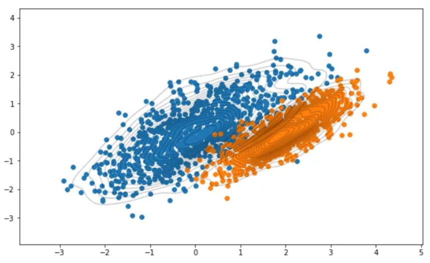


Figure 2.8: Clusters produced by the GMM algorithm [66]

indicate that a sample has been assigned to the wrong cluster, as a different cluster is more similar. The latter would generally indicate that something is wrong with the clustering algorithm.

$$\text{Silhouette} = \frac{1}{n}\sum_{1}^{n}\frac{b-a}{max(a,b)} \tag{2.6}$$

Another option is the Calinski-Harabasz (CH) score, which is also known as the Variance Ratio Criterion. This metric computes the ratio of the sum of between-cluster dispersion and within-cluster dispersion as shown in Equation 2.5.1. B is the between-cluster variance, and W is the within-cluster variance, both calculated with the sum of squares, while k is the number of clusters and n is the number of data samples. The objective is to maximise B and minimise W, and a higher CH score is therefore desirable.

$$\text{CH} = \frac{B*(n-k)}{W*(k-1)} \tag{2.7}$$

Although both the silhouette score and CH score indicate how distinct clusters are, they do so based on different criteria. While the silhouette score measures the distance between and within clusters, the CH score measures the dispersion. It can therefore be beneficial to evaluate clusters using both metrics.

When true labels for classes are available, other metrics can be used as well. Completeness measures how many data samples from the same class are in the same cluster, while homogeneity measures how many data samples in a cluster are from the same class. V Measure Score is the harmonic mean of the completeness and homogeneity, calculated as shown in Equation 2.8. The calculation for homogeneity and completeness is also shown, where C is the predicted class, K is the true class, and H is the entropy of a given group of samples.

$$\begin{aligned}\text{homogeneity(C)} &= 1 - \frac{H(C|K)}{H(C)} \\ \text{completeness(C)} &= 1 - \frac{H(K|C)}{H(C)} \\ \text{v} = \frac{2 * homogeneity * completeness}{homogeneity + completeness}\end{aligned} \tag{2.8}$$

Another supervised clustering metric is Adjusted Rand Index (ARI), which is based on the Rand Index (RI). The RI score measures the similarity between two clusters, considering all possible pairs of data samples. The pairs that are in the same or different clusters in the predicted and true clusters are then counted. This score is adjusted for chance, as shown in equation 2.9. The best score for this metric is 1, and it is lower bound to -0.5.

$$\text{ARI} = (RI - Expected\_RI)/(max(RI) - Expected\_RI) \tag{2.9}$$

In addition to metrics, visual inspection of clusters can also be advantageous. For high-dimensional latent representations, it is difficult to present the clusters. Therefore, dimensionality reduction techniques, such as Principal Component Analysis (PCA), can be of interest. PCA transforms vectors to a lower-dimensional representation, while still preserving the closeness of the original dimensions. This is desirable for evaluating clusters, as it allows the transformation of the latent space to a 3D space, which can be plotted and visualised easily. However, several things can make the transformed latent space collapse into one single cluster. One of these things is noisy samples. Brubaker [67] proposed to use Robust PCA to handle noisy samples, which was applied before clustering.

Metrics such as silhouette score and Calinski-Harabasz (CH) score show how compact the clusters are, and how separated different clusters are from each other. On the other hand, when clustering is performed on latent representations, these metrics do not demonstrate how this grouping translates to the raw input data. Metrics that are based on labels might not provide a good basis for evaluation either, as the clusters can represent something else than the labelled classes. It is therefore difficult to meaningfully evaluate these clusters. When dealing with this challenge, Zhao et al. [56] propose splitting driving scenes into separate samples, thus augmenting the dataset. By doing this, [56] create labels for which samples are known to be similar, as they are the same driving scenes, and can modify the loss function to force these samples closer together in the latent space. Multiple loss functions can be used for this purpose, one of them being contrastive loss, proposed by

Hadsell, Chopra and LeCun [68]. Contrastive loss is often used in SSL and minimises the distance between neighbouring samples while maximising the distance to samples that are further away. By doing this, instead of using a typical loss function employed in classification tasks such as CE loss, one does not need labels, only prior information to identify neighbours [68]. The computation of contrastive loss is shown in Equation 2.10. In this formula, x_ij is the measure of similarity between samples i and j, which can be the Euclidian distance, and τ is a temperature normalisation factor.

$$L_{i,j} = -log \frac{e^{x_{ij}/\tau}}{\sum_{k=1[k \neq i]}^{2N} e^{x_{ik}/\tau}} \tag{2.10}$$

Another loss function that can be used is triplet loss, proposed by Schroff, Kalenichenko and Philbin [69]. In previous research, it has often been used for training Siamese networks[5] [70] but has also been used for training VAEs [71]. Triplet loss, and other derived variations, have been used for a variety of clustering tasks in previous research [72, 73, 74]. It can be defined as shown in Equation 2.11, where $x_{a,n}$ is the distance between an anchor and a negative sample, and $x_{a,p}$ is the distance between an anchor and a positive sample. Negative and positive samples refer to whether a sample is deemed similar to the anchor sample from a priori knowledge. The last variable, m, is a margin value used to keep dissimilar samples far apart. Again, this minimises the distance in the latent space between similar samples and maximises the distance between dissimilar samples. The distance function can be the Euclidian distance. However, when the latent space is highly dimensional, all samples are deemed far apart. Therefore, using the cosine angle between latent vectors can be more efficient.

$$L_{i,j} = max(x_{a,n} - x_{a,p} + m, 0) \tag{2.11}$$

While the contrastive loss is greedy and tries to force samples from the same class to be in the same spot in the latent space, triplet loss converges to a point where samples from different classes are easily separated. In addition, triplet loss considers the margin value when considering samples from the same class, thus structuring the space within clusters as well. This can be beneficial for the goal of generalising latent representations and further organising the latent space.

Anomaly Detection Clustering has also been used as a method for anomaly detection. Li et al. [75] propose a clustering-based approach to detect anomalies in the amplitude and shape of multivariate time series data.

[5]Siamese neural networks apply the same weights to two inputs, allowing comparisons of the two resulting feature vectors.

Ouyang and Sanchez [76] propose a probability-based approach to clustering latent representations. The latent representations are generated by a deep denoising AE, and the method is used to cluster videos.

2.5.2 Supervised Evaluation Methods

In cases where labelled data is available, it is possible to evaluate latent representations using supervised methods such as classification. As latent representations are easier to handle than raw input data due to the decreased dimensions, several methods can be applied for classification tasks. Previously researched methods include GMMs, fine-tuned models, Support Vector Machines (SVM), and random forests [77]. Fine-tuning is a form of transfer learning, which is a technique where a model is first trained for one task with sufficient available data, and then trained for a similar task [78] where data might be limited. In their survey, Zhuang et al. [79] explore the many forms of transfer learning, in addition to describing successful applications in sectors such as medicine, bioinformatics, and transportation.

Chapter 3

Dataset

This chapter presents the dataset used for the reported experiments and is divided into three sections. **Section 3.1** describes the classes and other labels included in the dataset. It also describes how the dataset was annotated. **Section 3.2** describes the VMS data and how it is pre-processed. **Section 3.3** describes the video data and how it is pre-processed.

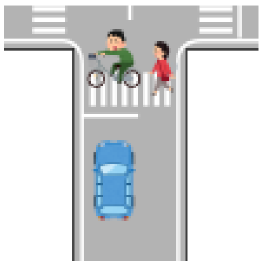
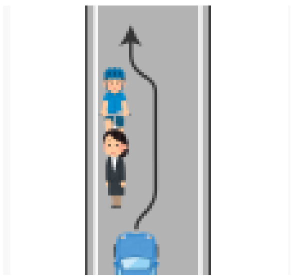
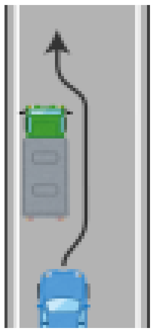
The NEDO dataset contains real-world city traffic scenes and is used for testing the implementations and hypothesis of this thesis. NEDO stands for National Research and Development Agency-New Energy and Industrial Technology Development Organization, and the dataset was collected by the organisation for an AI research and development project called "Development of next-generation artificial intelligence/robot core technology".

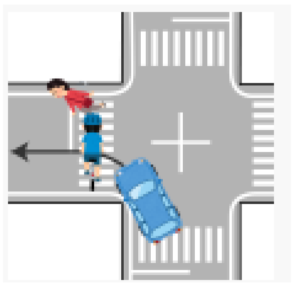
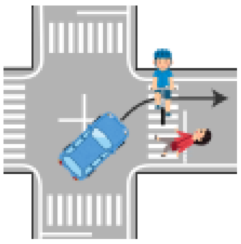
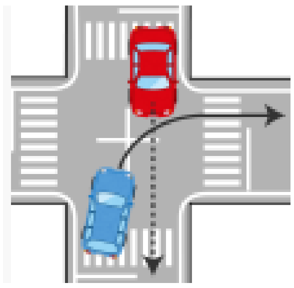
The reported experiments are tested using data collected in 2019. The first reason for only using one year out of the four available is that the annotation and gathering methods change each year, making it difficult to train a model using data from every year. The second reason is that only NEDO-2019 contains sensor data from Mobileye, which is an AI-based sensor, as mentioned in section 2.2. In short, the NEDO-2019 dataset contains 2685 driving scenes gathered around the Nagoya University campus, where each recording lasts approximately seven seconds.

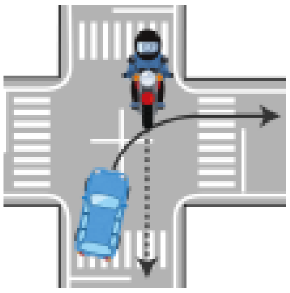
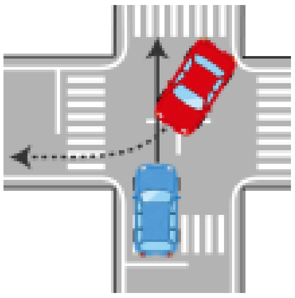
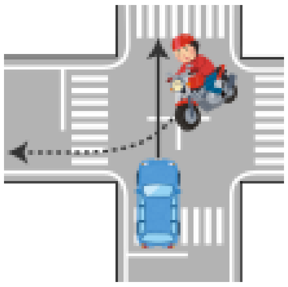
3.1 Classes and Annotation

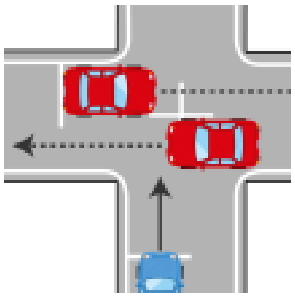
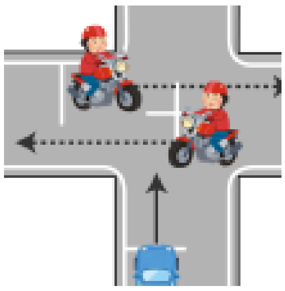
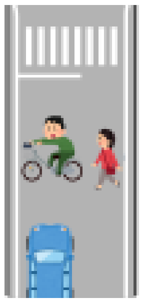
This section describes the different types of labels in the dataset, and how the dataset was annotated. The driving scenes are categorised into 14 classes, each of which is listed, illustrated, and explained in Table 3.1. The illustrations were made by the Behaviour Signal Processing Laboratory of Nagoya University.

Table 3.1: Illustrations and explanations of driving scenario classes

Class and illustration	**Explanation**
1. Pedestrian or bicycle crossing 	Single or multiple pedestrians/bicycles on a crosswalk in front of the ego vehicle. Pedestrians and bicycles more than 30m away are not covered.
2. Passing Pedestrians and Bicycles 	Pedestrians/bicycles on or next to the road, being overtaken by the ego vehicle.
3. Overtaking parked vehicles 	A parked vehicle on the street is overtaken by the ego vehicle.

Class and illustration	**Explanation**
4. Pedestrians and cyclists when turning left 	Ego vehicle entering an intersection to turn left, with pedestrians/bicycles in the scene. Pedestrian or bicycle does not need to be crossing the road.
5. Pedestrians and cyclists when turning right 	Ego vehicle entering an intersection to turn right, with pedestrians/bicycles in the scene. Pedestrian or bicycle does not need to be crossing the road.
6. Car facing straight when turning right 	Ego vehicle entering an intersection to turn right, with an opposing car going straight into the intersection.

Class and illustration	**Explanation**
7. Motorcycle facing straight when turning right 	Ego vehicle entering an intersection to turn right, with an opposing motorcycle, or multiple, going straight in the intersection.
8. Car turning right when going straight 	When the vehicle is about to go straight through an intersection, a car in the oncoming lane is turning right.
9. Motorcycle that turns right when going straight 	When the vehicle is about to go straight through an intersection, a motorcycle in the oncoming lane is turning right.

Class and illustration	**Explanation**
10. Car(s) crossing when going straight 	One or multiple cars crosses in front of the ego vehicle. Does not depend on whether the ego vehicle is stopped or moving.
11. Motorcycle(s) crossing when going straight 	One or multiple motorcycles crosses in front of the ego vehicle. Does not depend on whether the ego vehicle is stopped or moving.
12. Illegal Pedestrians and Bicycles 	Single or multiple pedestrians/bicyclists crossing the road, not in a crosswalk.

Class and illustration	**Explanation**
13. Near-miss denoted by the driver	A near miss, meaning close to an accident, is denoted by the driver. This can include driving scenes where a near miss happens that is difficult to observe from the video recording, as the driver might observe something the forward-facing camera does not capture.
14. Near-miss denoted by the annotator	A near miss, meaning close to an accident, is marked by the annotator.

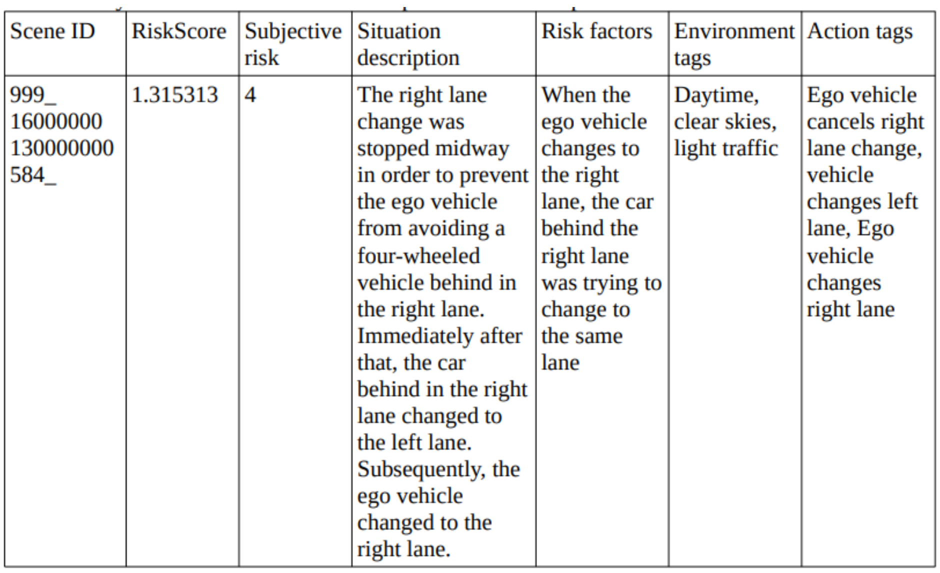
3.1.1 Annotation Process

13 annotators worked on the data set, all of which annotated a different number of samples. The annotation includes scene ID and class label, risk score, subjective risk, situation description, risk factors, environment tags, and action tags. An example of an annotated scene is given in Figure 3.1. The annotation is originally done in Japanese but is translated to English for this example. The different annotations are explained below.

1. Scene ID: ID for each scene of urban driving
2. RiskScore: Risk score calculated by post-processing
3. Subjective risk: Five levels of risk assigned to the scene by the operator (1: Safety => 5: Dangerous)
4. Situation description: Short sentence sequence that describes the driving situation
5. Risk factor: Short sentence string that describes the risk factor (only for scenes where the operator gives subjective risk levels of 4 and 5)
6. Environment tag: Static keyword that indicates the driving environment/situation

7. Action tag: Short sentence string that indicates the driving environment/status

In the data set, the class label is given by the last number of the scene ID: 999_16000000130000000584_x, where x denotes which of the 14 classes the sample belongs to. RiskScore is a score obtained by normalising the five levels of subjective risk of scenes assigned by each operator using Likert's sigma method [80], resulting in a single value that represents the combined assessment of all annotators.



Scene ID	RiskScore	Subjective risk	Situation description	Risk factors	Environment tags	Action tags
999_ 16000000 130000000 584_	1.315313	4	The right lane change was stopped midway in order to prevent the ego vehicle from avoiding a four-wheeled vehicle behind in the right lane. Immediately after that, the car behind in the right lane changed to the left lane. Subsequently, the ego vehicle changed to the right lane.	When the ego vehicle changes to the right lane, the car behind the right lane was trying to change to the same lane	Daytime, clear skies, light traffic	Ego vehicle cancels right lane change, vehicle changes left lane, Ego vehicle changes right lane

Figure 3.1: Example of driving scene annotation

3.1.2 Imbalanced Class Distribution

As seen in Figure 3.2, the dataset does not contain an equal number of samples for each of the 14 classes. However, the uneven distribution is not significant, and this distribution will suffice for this report. The two classes that are noteworthy in terms of the number of samples are class 7 when a motorcycle is facing straight and the ego vehicle is turning right and 9 when a motorcycle in the oncoming lane turns right and the ego vehicle is going straight. This class imbalance is something to be mindful of when evaluating label classification performance. It is outside the scope of this report, but had it been a significant difference in the number of samples in each class, data augmentation could have been an option. As Wang et al. [81] show, regular data augmentation techniques often do not work as well for driving-related images, such as mirroring an image. Regular data augmentation techniques often result in images where the driver sits on the wrong side of the car, or where cars drive on the wrong side of the road. Another option would be the VideoMix algorithm proposed by Yun et al. [82], where they place a video within another video, thus creating a new, noisy data sample. This algorithm shows promising results in their paper. In their paper, Zhu et al. [83] propose an algorithm to detect dangerous driving behaviour in a data set with a small proportion of dangerous drivers. Drawing inspiration from [83] could be an option if mainly focusing on predicting risk scores. However, as this thesis includes working with both video and VMS data, generating new data for data augmentation is a difficult and complex task. Since the data is sequential, splitting the sequences into multiple data samples is a more viable option, thus keeping the relation between modalities consistent.

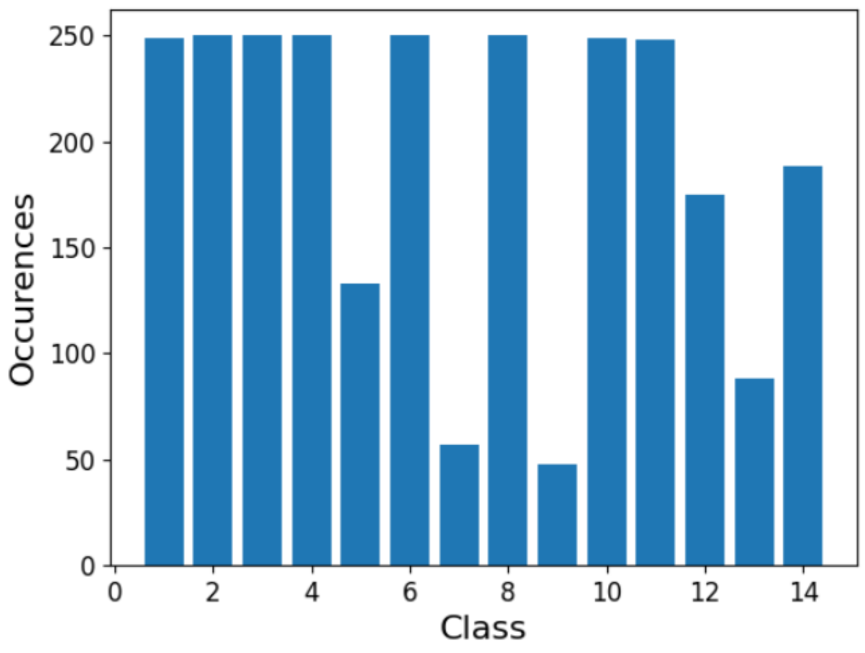


Figure 3.2: Distribution of classes in dataset

3.2 Vehicle Monitoring Sensor Data

One of the modalities used in this dataset is a collection of VMS data that describe different aspects of the driving scenes. The VMS data comes from four different sensors and is given as sequential numerical data. The first three sensors are GPS, CAN, and recorded driver information, of which there are 74 unique features. The recorded driver information describes the driver, such as whether their eyes are open, their heart rate, and where their head is facing. The information from the other two sensors has an overlap in features, such as those describing velocity and acceleration. In addition to these features, there is also data from the AI-based Mobileye sensor, which contains 19 features describing the ego vehicle and environment, 16 features for each car that appears in the driving scene, and 7 features for each pedestrian and bicycle in the driving scene. There is also some overlap between the Mobileye and GPS features, such as lane information and yaw and pitch of the ego vehicle.

3.2.1 Selecting Features

Before using this modality, it is important to analyze the features to review contributions, correlations, and anomalies. This process is described in this subsection, where some features are removed based on the results.

To start, there are quite a few features that only contain values of zero. This can come from a fault in the sensor during recording, or from the similarity of the driving scenes, causing the value to be the same for every scene. Some categorical features also use zero as "unknown", which can be the reason as well. These features will not be used for the experiments of this report. There are some features, such as the ID of objects and other formalities, which are also removed before using the VMS data.

K Nearest Neighbors A common method for feature analysis and selection is the K Nearest Neighbors (KNN) algorithm, which is non-parametric and uses labels to assign the class of a sample. The algorithm works by choosing the majority class of the K nearest neighbours of a sample. Although it is quite computationally expensive and takes a long time to run, it requires no training. In addition, the error rate is upper bound to twice the optimal error rate, which gives a good estimate of how well a given feature will perform. However, given poor initial test results, no features are removed based on this test.

Correlation Another approach to selecting features is to calculate the correlation between each feature. As mentioned previously at the beginning of section 3.2, the GPS and vehicle sensors provide some similar features. Ideally one would choose the sensor that provides the most accurate data for each feature, but that is difficult to do in practice. It is especially important to remove features that are present in two or more sensors, as it would give too much weight to that information. However, perhaps due to the varying format of different sensor inputs, the results of this test are inconclusive. Therefore, it is not possible to reasonably remove features based on the calculated correlation. Some features are however removed based on manual analysis, as they essentially provide the same information as other features.

Selected features Table 3.2 lists and describes the features which are used in the reported experiments.

Table 3.2: Description of VMS data features

Feature	**Description**
	From sensor: Driver
Eye-opening rate	The openness of the driver's eyes given in millimetres
Face pitch	The forward/backward tilt of the driver's head, given in degrees
Face yaw	The left/right direction the driver's face is pointed, given in degrees.
Horizontal gaze	The horizontal direction the driver is looking, given in degrees.
Vertical gaze	The vertical direction the driver is looking, given in degrees
Heart rate	Heart rate of the driver, given in bpm
	From sensor: GPS
Distance from the last intersection	Given in centimetres
Distance to last intersection	Given in centimetres
Road curvature	Given in degrees
Intersection	1 if scene contains an intersection, 0 if not
Traffic light	1 if scene contains a traffic light, 0 if not
Lanes	Categorical feature indicating which of 11 types of lanes is in the driving scene
Road type	Categorical feature indicating which of 63 types of road is in the driving scene
Road width	Categorical feature indicating which of 11 classes of road width is in the driving scene
Vehicle turn state	A categorical feature which indicates if the vehicle is turning right, left, or not at all
Vehicle movement	Vehicle velocity, given in km/h
Vehicle pitch angle	The forward/backward tilt of the car, given in degrees
Vehicle yaw rate	The left/right orientation of the vehicle, given in degrees
	From sensor: Mobileye
Mobileye 2	Lane marker type on the left side, 7 classes
Mobileye 3	Lane marker type on the right side, 7 classes

Feature	**Description**
Mobileye 4	Distance to the left lane marker, given in meters
Mobileye 5	Distance to the right lane marker, given in meters
Mobileye 6	The speed limit in the driving scene, given in km/h
Mobileye 7	The pitch angle of the vehicle, given in radians
Mobileye 8	Yaw angle of the vehicle, given in radians
Mobileye 9	Road curvature, given in 1/m
Mobileye 10	Road curvature detection, 0: unknown, 1: R $<$ 100, 2: 100 $<$ R $<$ 300, 3: 300 $<$ R $<$ 500, 4: 500 $<$ R $<$ 700, 5: 700 $<$ R $<$ 1000, 6: 1000 $<$ R
Mobileye 11	Lane change information, 0: unknown, 1: right-side change, 2: left-side change
Mobileye 14	The direction of the vehicle ahead, 0: Not Approaching, 1: Approaching
Mobileye 15	Status of approaching vehicle, 0: unknown 1: Approaching from left 2: Approaching from the right
Mobileye 16	Distance to car ahead, given in meters
Mobileye 17	The velocity of the car ahead, given in m/s
Mobileye 18	Time to collision (TTC), given in seconds
Mobileye 19	Pedestrian/bicycle amount in driving scene
Mobileye: observed cars 2	Neighbouring object longitudinal position (forward), given in meters
Mobileye: observed cars 3	Neighbouring object latitudinal position (horizontal), given in meters
Mobileye: observed cars 4	Lane change information of a neighbouring vehicle, categorical feature
Mobileye: observed cars 5	Blinker information of a neighbouring vehicle, categorical feature
Mobileye: observed cars 6	Relative Speed to a neighbouring vehicle, given in m/s
Mobileye: observed cars 7	Neighbouring vehicle type, regular car, truck, motorcycle or unknown
Mobileye: observed cars 8	Movement state of neighbouring vehicle
Mobileye: observed cars 9	Neighbouring vehicle brake light status, 0: off, 1: on
Mobileye: observed cars 10	Neighbouring vehicle length estimation, given in meters

Feature	**Description**
Mobileye: observed cars 11	Neighbouring vehicle width estimation, given in meters
Mobileye: observed cars 12	Number of detections
Mobileye: observed cars 13	Cruising Lane, 0: not assigned, 1: ego lane, 2: different lane 3: invalid signal
Mobileye: observed cars 14	Neighbouring vehicle speed, given in m/s
Mobileye: observed cars 15	Relative Acceleration, given in m/s^2
Mobileye: observed cars 16	Neighbouring vehicle eccentricity (yaw), given in degrees
Mobileye: observed pedestrians/bicycles 2	Pedestrian/Bicycle detection, 0: Unknown, 1: Pedestrian detected, 2: Bicycle detected
Mobileye: observed pedestrians/bicycles 3	Pedestrian/bicycle distance, given in meters
Mobileye: observed pedestrians/bicycles 4	Pedestrian/bicycle distance lateral position, given in meters
Mobileye: observed pedestrians/bicycles 5	Pedestrian/bicycle TTC, given in seconds
Mobileye: observed pedestrians/bicycles 6	Number of detections
From sensor: CAN	
Linear acceleration x, y, z	The acceleration of the ego vehicle in directions x, y, and z, given in m/s^2
Speed pulse	The original pulse signal from which the vehicle velocity is calculated
Brake pedal hydraulic	The hydraulics of the brake pedal, given in Mpa
Gas pedal position	Percentage of how much the gas pedal is pressed
Speed	The velocity of the ego vehicle, given in km/h
Steering angle	The angle of the steering wheel, given in degrees
Back signal	Indicates whether the back signal of the ego vehicle is on or off
Brake signal	Indicates whether the brake signal of the ego vehicle is on or off
Seat belt state	Indicates whether the seat belt of the driver is used
Turn signal	State of ego vehicle's turn signal
Headlight state	State of ego vehicle's headlight state

3.2.2 Sample rates

Each of the 4 sensors has different sample rate ranges. Combined with a slight difference in the length of each driving scene, there is a large variation in the number of data points across and within features of a data sample. Figures 3.3 and 3.4 show the mean and standard deviation of the number of samples of each feature respectively and demonstrate the large variation in the number of data points. This leads to varying shapes of the inputs, which is a problem for most ML models. To remedy this, the data is resampled to a fixed number of timesteps, using interpolation for continuous features and copying the previously observed value for categorical features. To find a good number of timesteps, the MSE between the original and resampled values at equal timesteps is calculated. However, this method only measures the difference in values and does not account for a shift in the timestamps. Therefore the correlation between the original and resampled values is also calculated. As seen in Table 3.3, 256 appears to be a reasonable number of samples as the MSE and correlation values do not decrease as much when increasing the number of timesteps further.

Table 3.3: Mean Square Error (MSE) and correlation for resampled time series, averaged over all features from data based on 2000 random samples

N samples	**MSE**	**Correlation**
50	83112890652	0.89
100	3.115	0.98
150	2,282	0.987
200	2,046	0.99
300	1,098	0.997
500	0	0.999

In addition, Figure 3.3 shows that not many features contain more than 256 data points. Keeping in mind that increasing the number of timesteps would also increase computational cost, 256 samples seem reasonable. The number 256 is chosen instead of 250, as it is a binary number, and therefore easier to work with in an ML context.

As seen in Figure 3.5, where blue crosses mark the resampled values and red dots mark the original values, the overlap of resampled and original data points is quite good. Note that the x-axis shows timesteps and not sample numbers. The number of timesteps is larger than the number of samples, due to the slight difference in timestamps of original and resampled data points. However, even though the new number of samples is larger than the original number of samples, a certain period might contain more data points, leading to down-sampling in that specific time frame. The figure also indicates that there might be missing data points in a sequence. This is also supported by the standard deviation of the number of data points as shown in Figure 3.4,

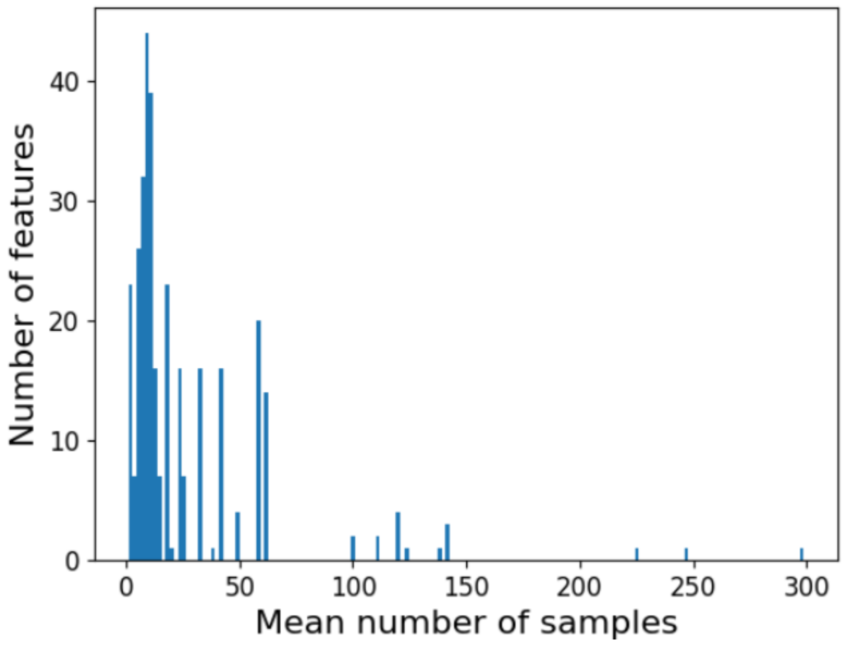


Figure 3.3: Mean number of samples for each feature

which is quite high given the small variation in the length of different driving scenes. These missing data points are also filled by either interpolation or mean values for continuous and categorical variables respectively.

3.2.3 VMS Data Pre-Processing

There are several things to consider before using the VMS data. The data contains output from several sensors, so they must be treated differently. This subsection describes how the data is normalised, and how the different types of data are handled.

Normalisation The VMS features are given in different metrics including km/h, m/s, radians, and degrees. Some features, such as relative velocity, can have negative values, while others can not. For these reasons, normalisation is applied to the features as part of the pre-processing stage. Min-max normalisation is a common technique, which scales all values between 0-1, and the calculation is shown in Equation 3.1. This results in giving equal importance to each feature, regardless of the value ranges. However, there is a question of which min and max values to apply. The sensor for the velocity of the vehicle has a range between 0 and 255, as has several other features. However, as most of the driving scenes are recorded in urban areas, the ve-

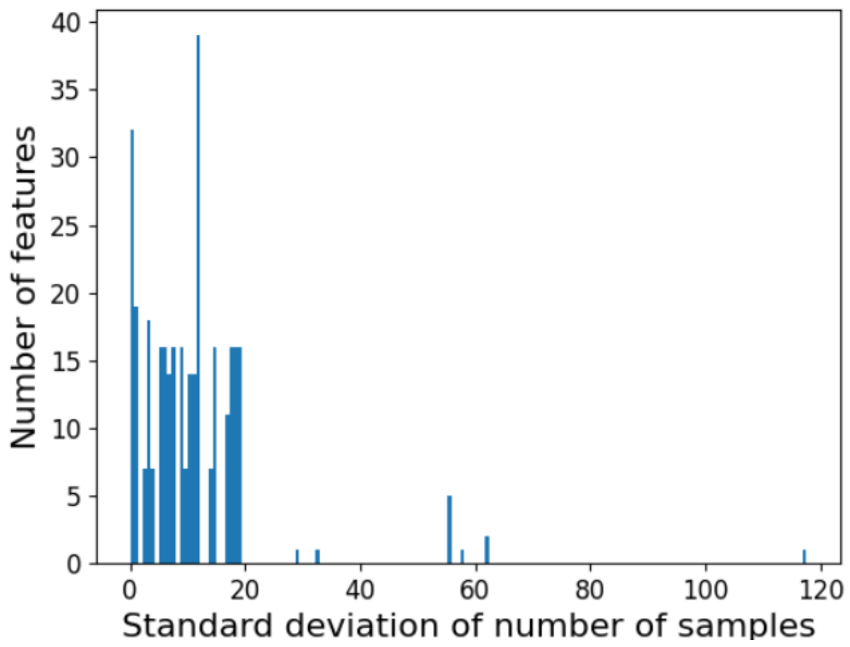


Figure 3.4: Standard deviation of the number of samples for each feature

locity seldom exceeds 100 km/h, and the max velocity found in the dataset is 120 km/h. For this reason, using minimum and maximum values from sensor documentation for min-max normalisation is not the best alternative. Therefore the minimum and maximum values found in the dataset are used for min-max normalisation. Another option would be to use z-score standardisation, which handles outliers better than normalisation. As with the resampling procedure, it is important to handle categorical features differently, which need not be normalised.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

Categorical features Some of the categorical features can be either 0 or 1 and do not need further handling, while others have a higher number of categories. The latter type of categorical features can be handled by using embeddings. Using learned embeddings is more computationally effective than one-hot-encoding[1] [84, 85]. Using one-hot-encoding would lead to a very large input dimension, and thus too many trainable parameters. Meanwhile,

[1]One-hot-encoding uses an array with one value describing the probability of each category.

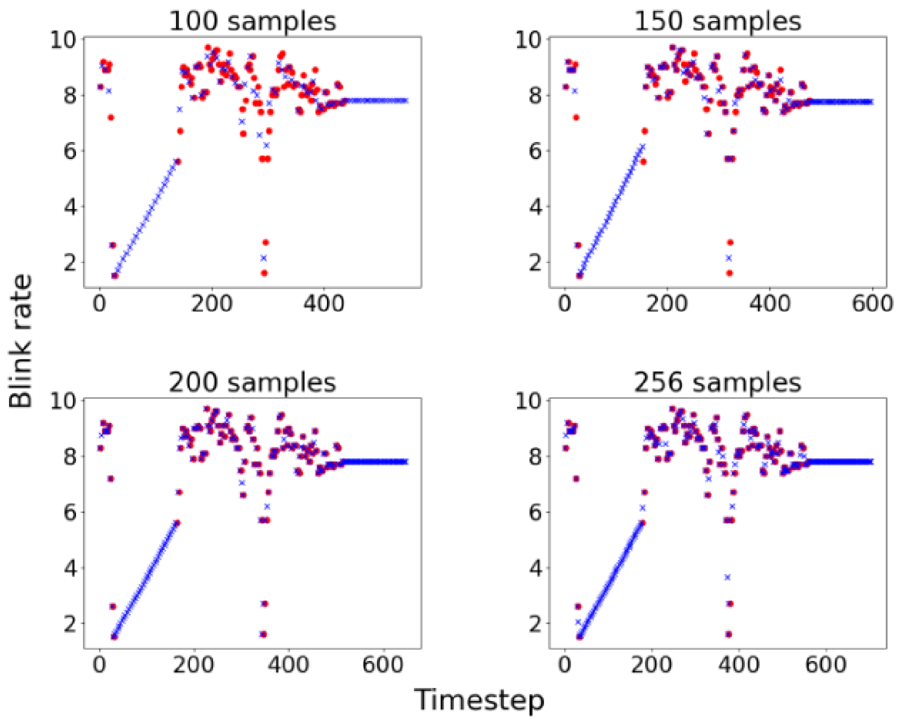


Figure 3.5: Resampled data points of "blink rate" feature

embeddings transform the 1-dimensional feature vector to a 2-dimensional vector of a given size, while preserving semantic relationships. This means that similar categories are represented close to each other in the embedded vector space. While not applied in this thesis, an option is to vary the size of each embedding based on the number of categories in the feature.

The categorical features are already encoded with integer encoding, meaning that instead of the category label, an integer denoting the label is given. Although this is possible to use as input for an ML model, it is not optimal. Integer encoding implies similarity between categories when the integers are close, which is not necessarily the case.

Missing features Of the 67 chosen features, the 47 that are not specific to detected objects should in theory be available in all data samples. However, only a handful is available in all of them, as shown in Figure 3.6. This poses a problem as each sample will have a different number of features, again leading to varying input sizes. This processing step is quite simple, as it just involves adding the missing features using the masking value -99.

It is important to make sure that each feature appears in enough samples to be able to properly train the model parameters relating to that feature.

As shown in Figure 3.6, there is a big spread in the number of appearances of different features. However, all features with lower than 1700 appearances are specific to detected objects, which is explained in the next paragraph. It is recommended to have at least 5 data samples per feature [86], and the features describing the ego vehicle have enough data to be properly trained.

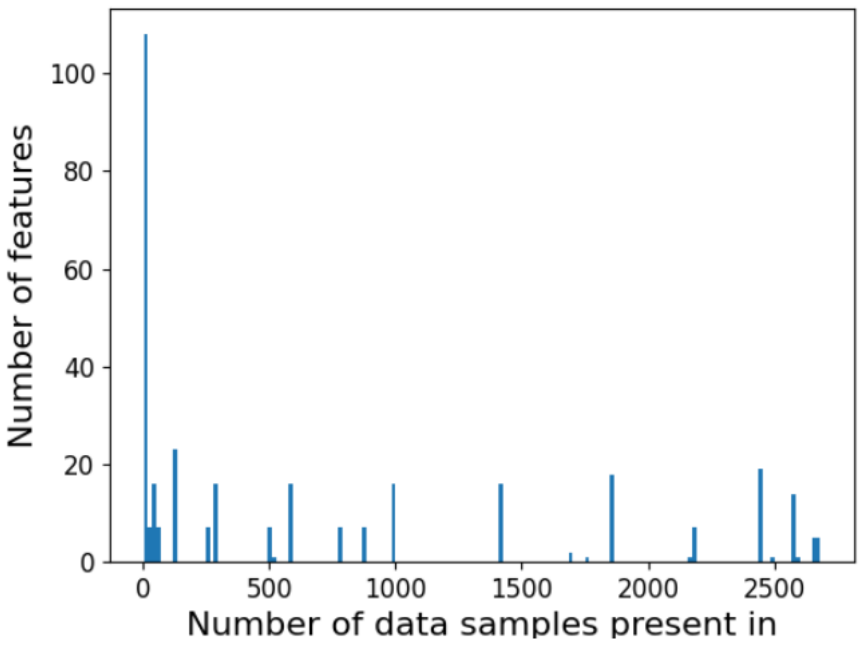


Figure 3.6: Distribution of the number of data sample appearances for features

Features for detected objects These are features that describe either a vehicle, pedestrian, or bicycle observed in the driving scene. Handling these features presents a similar problem to the one discussed in the previous paragraph. Although there are no missing features for an object if it is detected, the number of objects varies across data samples. This leads to practical problems when using a data loader[2] with a batch size, as all samples in the batch need to have the same dimensions. Again, this can be solved by adding features, this time until all samples in the batch have the same number of features. It is important to use a value that separates these features from others, and -99 is chosen for this purpose.

[2]A data loader is a coding implementation that fetches data from a dataset and often fetches it in batches.

As mentioned at the beginning of section 3.2, there are 16 different features describing each of the neighbouring cars that appear in the driving scenes, and 7 different features describing each choosing pedestrian or bicycle in the detected object features. The highest recorded number of neighbouring cars in a driving scene is 12, and 11 for pedestrians/bicycles. This results in a high number of features if they were to be handled as individual features. Handling them as multiple instances of the same features would drastically lower the number of features, and lower the number of parameters needed in a model. Importantly, many of these features do not appear many times in the dataset, meaning that the weights involved would not be trained properly. This is due to the varying number of cars, pedestrians, and bicycles in each driving scene. Therefore, treating them as the same features will most likely lead to higher performance, even though it might provide some additional difficulties when setting up the model structure. Another option would be to set a cut-off point for minimum appearances of a feature, and only use those that would allow for proper training of corresponding weights. An issue with this approach is that the available information is not used to its full extent.

Presenting data for models Before using the data for the models, it is split into eight groups:

1. Continuous standard[3] features
2. Binary standard features
3. Categorical standard features
4. Continuous features describing pedestrians and bicycles detected in the driving scene
5. Categorical features describing pedestrians and bicycles detected in the driving scene
6. Continuous features describing cars detected in the driving scene
7. Binary features describing cars detected in the driving scene
8. Categorical features describing cars detected in the driving scene

Note that there are no binary features for pedestrians and bicycles. The reason for splitting the features into continuous, binary, and categorical is for practical reasons, making it easier to know which features must be handled differently regarding normalisation and embeddings.

[3]These are features from all the sensors that describe the ego vehicle and its surroundings.

3.3 Video Data

The cars are equipped with a front-facing colour camera, that records a video for each driving scene. The ego vehicle is not visible in the driving scene. Each video is about 7 seconds long and has a frame rate of 26 frames per second. Shown in Figures 3.7 and 3.8 are frames taken from two separate videos.

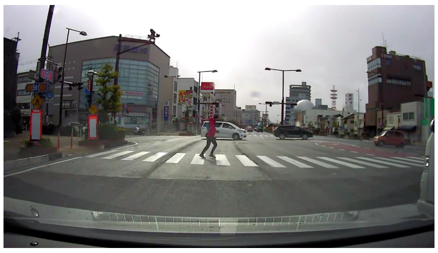


Figure 3.7: Frame from video of class 1

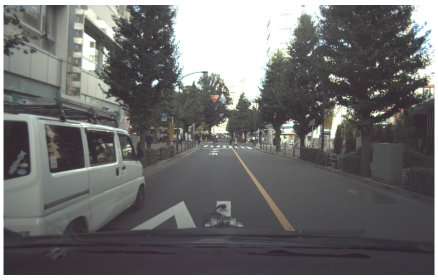


Figure 3.8: Frame from video of class 3

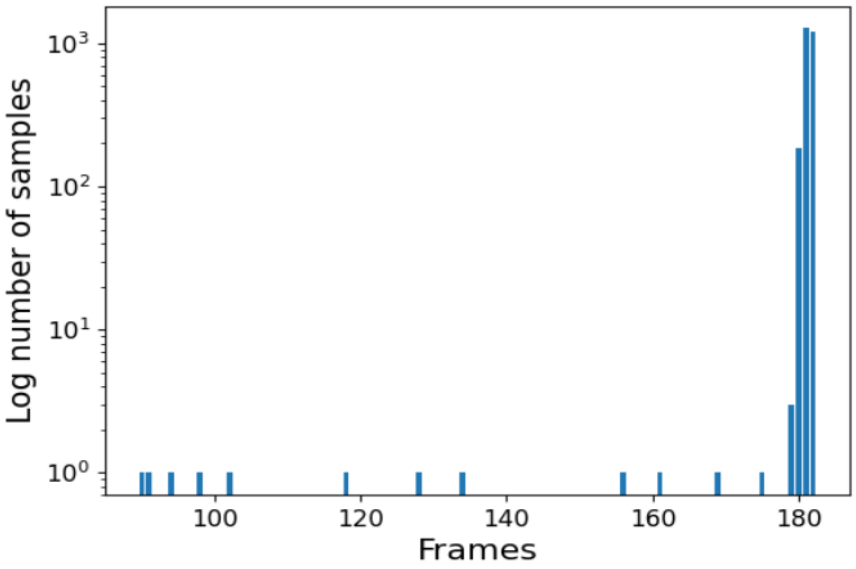


Figure 3.9: Logarithmic distribution of the number of frames in videos from the dataset

Frame rate However, as mentioned in subsection 3.2.2, some driving scenes are not precisely seven seconds long. As shown in Figure 3.9, most of the videos contain a similar number of frames. Note that the figure shows the natural logarithm of frame rates, to better visualise the smaller values. 99% of the videos contain 180, 181, or 182 frames. As mentioned in subsection 3.2 regarding the time series data, most ML models need a fixed input size, so the variation in frames can pose a problem. As a frame rate of 26 fps can be excessive regarding computational expense, 5-10 fps should be enough. An alternative is to choose a fixed, smaller number of frames. This means that the frame rate will vary across samples, but given the distribution of frames in the samples this should not be significant. The videos are therefore resampled to contain 64 frames, thus refraining from up-sampling any videos and resulting in 9 fps.

Further processing As the videos contain values from 0 to 255 for each RGB channel, it is uncomplicated to normalise the values before training the model. There is also the issue of choosing a reasonable size for each frame and finding a middle ground between information loss and computational complexity. The complexity of the problem decreases as the frame rate and image size decrease, but the latent space might not be as useful for separating the videos into classes if the video quality is too low. The images shown in Figure 3.10 show frames from two videos that are resized to 256x256 pixels.

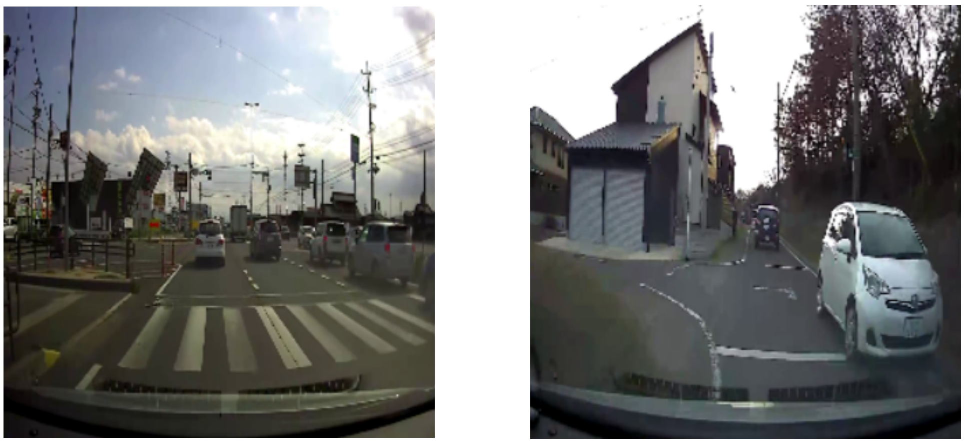


Figure 3.10: First frame of two separate resized videos from dataset

Chapter 4

Feature Extraction and Evaluation

This chapter describes the proposed methods of this thesis. **Section 4.1** explains how and why data augmentation is performed on the dataset. In **section 4.2** the encoder and decoder parts depicted in Figure 4.1 are described, and section **4.3** explains concepts and considerations regarding model training. Section **4.4** describes the evaluation tasks: clustering and classification.

Figure 4.1 illustrates the full process of the experiments. The encoder takes video and VMS data and produces latent representations. The representations are first decoded to reconstruct the original input, and the model is trained to optimise this objective. Note that for the models that only use one of the modalities, the process is the same, but with only one of the inputs, and one of the reconstructed outputs. After the model training is complete, the latent representations are evaluated in the clustering and classification tasks.

4.1 Data Augmentation

An important preparation step in this thesis is data augmentation. As mentioned in subsection 3.1.2, this can be done by splitting each sequence into equal parts. Given that the sequences are approximately 7 seconds each, an appropriate number of splits is four new samples. Any less, and there would not be much to gain from the data augmentation. Any more, and the sequences would be too short. Although there is enough data in the dataset to properly train the models, augmentation of the dataset has the following benefits:

- **More data** available. This allows for better training of the models.
- **Less complex** tasks. As the sequence is shorter, there are fewer features to extract. This can make it more likely to achieve good results.

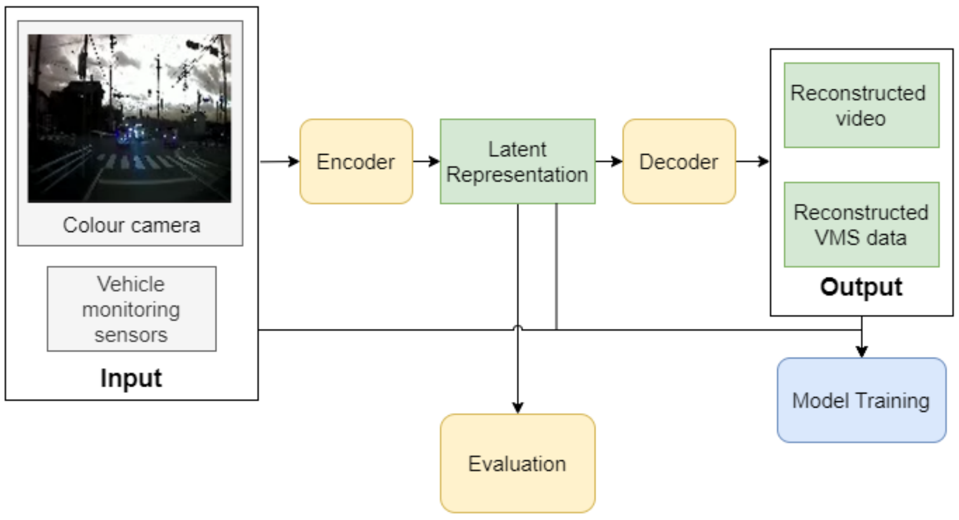


Figure 4.1: Flowchart of full experiment process

- **Creation of labels** for similarity. It is now possible to determine similarity, as two samples that are from the same sequence are more similar than two that are not.

The latter is the most significant benefit and is the reason behind performing data augmentation. This makes it possible to use triplet loss when training the models. Applying triplet loss will to some extent ensure that similarity in the latent space can be interpreted as similarity in input data. This makes the clustering results more meaningful and reliable, thus allowing the evaluation of the latent space with clustering.

4.2 Model Architectures

This section describes the architecture and implementation of the different models that are tested in this report. The models are implemented using the Pytorch library [87].

Figure 4.2 illustrates the training process of the models, and subsection 4.2.5 further explains the depicted loss functions. In short, the training objective is to reconstruct the given input. The other components of the loss function are forms of regularisation to prevent overfitting the models to both the training set and the reconstruction task. The loss function is designed to generalise the latent representations for a variety of tasks. The main components of the encoders can be described as:

- **Feature extraction**. Depending on the input data, this can be fully connected layers, convolutional layers, sequential layers, or a combination.

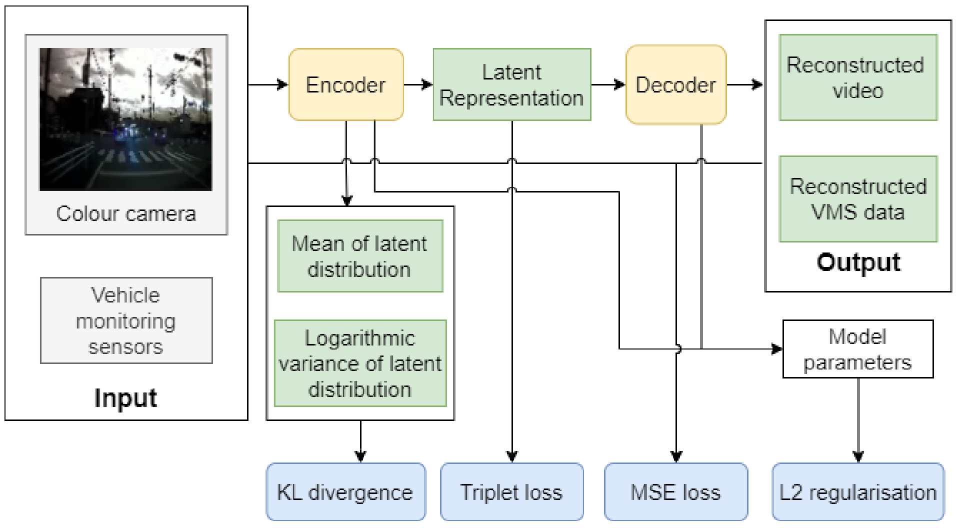


Figure 4.2: Flowchart of training process

- **Sequence compression**. Transforms the sequential data to a 1-dimensional vector.
- **Final layer**. The final fully connected layer produces either the latent representation or the mean and logarithmic variance to the distribution the representation is sampled from.

The specifics of the feature extraction component depend on the model. As for the decoders, they are essentially reverse copies of the encoder.

To address RQ1 from section 1.2, several model variations are implemented. By comparing a regular and a variational AE for the video data, the effect of incorporating KL into the loss function is studied. Further, another model is implemented to test the effect of 3D convolution vs. 2D convolution paired with a transformer layer. For VMS data, the effect of using GRU layers vs. transformer layers is researched. For RQ2 three different ways of combining modalities are tested: attention layers, high-level concatenation, and mid-level concatenation.

4.2.1 Video Autoencoders

In this segment, the structures of the Video AEs utilised in this thesis will be described. Three different variations are implemented and tested:

- **Video AE**
- **Video VAE**
- **Video Transformer VAE**

Video AE and Video VAE The first two implementations are quite similar. They both consist of four stacked 3D CNN layers in the encoder, which reduce the height, width, and number of frames of the input video. This method performs feature extraction and sequence compression concurrently with 3D convolution. The extracted features are then flattened to a 1-dimensional vector and passed through a fully connected layer. The difference between the two implementations is in the last layer of the encoder. The Video AE has an additional fully connected layer, which outputs the final latent representation, while the Video VAE has two separate fully connected layers. One of these outputs the mean of the distribution of the latent representation, and the second outputs the logarithmic variance. These outputs are then used to calculate the KL divergence as shown in Equation 2.5, which shapes the latent representation distribution. A latent representation is then sampled from this distribution and used for the reconstruction task. The encoder structure for both variations is depicted in Figure 4.3. The illustration includes vector fusion with the VMS modality, which is not the case for these models.

Video TVAE The last implementation for the video modality is the Video Transformer VAE. Again, stacked convolution layers are used to extract features from each frame. As opposed to the two aforementioned implementations, this one does not decrease the number of frames in the convolutional layers. The resulting sequential output contains the extracted features from the video, which is used as input for the transformer encoder layer. For the sequence compression stage, the output from the transformer encoder layer is flattened into a 1-dimensional vector, which is passed through a fully connected layer. As with the Video VAE model, the final layer is a two-part output, providing the mean and logarithmic variance of the latent representation. The structure of the encoder is illustrated in Figure 4.4.

For this model, the transformer decoder layer takes two additional inputs: memory and mask. The purpose of the memory in the transformer decoder layer is to provide information from the encoder about the input sequence, which is used by the decoder to generate the output sequence. The mask controls the flow of information in the transformer during training so that it can only access relevant parts of the input.

4.2.2 Vehicle Monitoring Autoencoder

Two variations for vehicle monitoring data feature extraction are implemented and tested in this report:

- **Vehicle Monitoring Sensor VAE**
- **Vehicle Monitoring Sensor Transformer VAE**

The reason for not testing a regular AE for the vehicle monitoring data, is the missing features discussed in subsection 3.2.3. Based on previous work described in subsection 2.4.1, Variational Autoencoders are a better fit for this task, as they do not encode the raw data directly to a latent representation, but rather to a probable representation sampled from a distribution. This means that even though the values for certain features are masked, the VAE should be able to extract the most probable feature information based on other features.

As outlined in section 3.2, managing the data from vehicle monitoring is a complex task. There are numerous factors to take into account when implementing the VMS AE models.

Missing Features Firstly, the missing features described in subsection 3.2.3 are handled. They are identified, added, and masked with the same value of -99. This value was chosen because real values are usually positive due to normalisation. While embedded values can be negative, they are typically close to 0. This approach indicates that the value is not real and that the true value should be based on a learned probability and context from other features. Another option is to fill missing features with the mean values from the batch, but this approach could hinder learning latent representations that differentiate between various driving scenes and would make data samples more similar.

Multiple Detected Objects As explained and shown in 3.2, there is a set of features for each detected vehicle and pedestrian/bicycle that appears in the driving scene. The maximum number of vehicles detected in a scene is 12, and the maximum number of pedestrians or bicycles is 11. The sets of features that describe the first vehicle and the first pedestrian/bicycle appear in almost all data samples. However, the number of appearances decreases as the number of detections increases, and the last sets of features only appear in a handful of samples. Due to this, the model weights connected to these inputs won't be properly trained. A solution to this is to reuse the same parameters for each object, thus allowing the parameters to be trained thoroughly. However, because of the varying number of objects in each data sample, this leads to the problem of varying dimensions. The output must therefore be combined in some way, into a set dimension.

More importantly, this might be a problem in the decoding part of the model. As there is an encoder and decoder for each feature, but not for each set of features for each detected object, the same encoder and decoder will be reused for each instance of detected cars and pedestrians/bicycles. This should be fine for the encoder, as each instance will have a different input. For the decoder, however, each instance takes the same input, the latent vector, and has to provide different outputs from that.

Loss for missing features As there is no way of knowing what the original data sample looks like for missing features, these are excluded from the loss calculation[1]. Nevertheless, it is desirable to train the model to decode these features properly, thus enforcing the latent representation to include this information. Therefore a random number of recorded features are masked and still included in the loss calculation. The pseudo-code for this is shown in Algorithm 1, which is structured so that it is highly probable that at least one feature in one of the data samples is masked, with a maximum of five features being masked. The code sets the full sequence of one feature, of one of the samples in the batch, to the mask value -99. There is a lot of room for optimising the probabilities and the maximum number of masked features. However, seeing as reconstructing missing features is not the main goal of the thesis, the results achieved with this method are adequate. The original feature tensor from Algorithm 1 is one of the eight split parts of the batched dataset. A random batch and feature are chosen, and the full sequence for that batch and feature is masked by -99, denoted by $masked_tensor[batch_idx, :, feature_idx]$. If the random batch and feature is a missing feature, and already masked as -99, new indices are randomly chosen.

Algorithm 1 Pseudo code for random masking of features

```
 1: $masked\_tensor \leftarrow$ original feature tensor
 2: for p in [0.9, 0.7, 0.5, 0.3, 0.1] do
 3:   $p \leftarrow$ p scaled by the number of features in input
 4:   $random\_number \leftarrow$ 1 with probability p, else 0
 5:   if $random\_number = 1$ then
 6:     $batch\_idx \leftarrow$ random batch index
 7:     $feature\_idx \leftarrow$ random feature index
 8:     while masked_tensor[batch_idx, :, feature_idx] $= -99$ do
 9:       masked_tensor[batch_idx, :, feature_idx] $\leftarrow -99$
10:       $batch\_idx \leftarrow$ random batch index
11:       $feature\_idx \leftarrow$ random feature index
12:     end while
13:   end if
14: end for
15: return $masked\_tensor$
```

Implementation This implementation concatenates the input into three different parts:

- **Standard features, describing ego vehicle and surroundings**

[1]This is the case with the padded detected object features as well.

- **Features describing detected pedestrians and bicycles**
- **Features describing detected neighbouring vehicles**

This allows the model to reuse the trained parameters for the different detected objects, which would not have worked had the model used the full input combined. By only splitting the input into three different parts, the model benefits from the added context of using multiple features. It is also more computationally effective, as opposed to encoding each feature individually. However, there is a downside to this implementation as well, arising from the practical side of implementing an ML model. As the size of the input is fixed, the input includes the masked missing features, in addition to the padded detected object features. These will then influence the encoded latent vector to an unknown degree.

Vehicle Monitoring Sensor VAE The VMS VAE is the first of the two VMS Autoencoders and is illustrated in Figure 4.3. This model uses a GRU layer to extract features from the standard features, and each detected object individually. Each resulting sequence is then compressed to a 1D vector by reshaping. This is followed by a fully connected layer, then two additional fully connected layers that output the mean and logarithmic variance of the latent representation distribution. After the process, the encoded standard features, and encoded features for a varying number of neighbouring vehicles, pedestrians, and bicycles are obtained. All encoded vectors for neighbouring vehicles are then summed together, and the same is done for encoded vectors for pedestrians and bicycles. Now there are three sets of mean and logarithmic variance vectors, one describing the ego vehicle and the surroundings, one describing neighbouring vehicles, and one describing pedestrians and bicycles. These are combined using an attention layer, which is trained to include the most useful information from all three sets of vectors. Ideally, an attention layer would be used to combine all features describing individual detected objects. However, because of the varying number of objects, this is difficult to implement.

VMS Transformer VAE The architecture of the VMS TVAE is slightly different. Here, the standard features, and detected object features are passed through separate fully connected layers, to obtain the same dimensions. Then, as with the model described above, the individual object features are summed together, to obtain three sequential vectors for standard features, neighbouring vehicle features, and pedestrian and bicycle features. These are combined using an attention layer. The single vector is then processed by a transformer encoder layer and finally passed through fully connected layers to obtain the mean and logarithmic variance of the latent representation distribution. This implementation is illustrated in Figure 4.4, only without the video components.

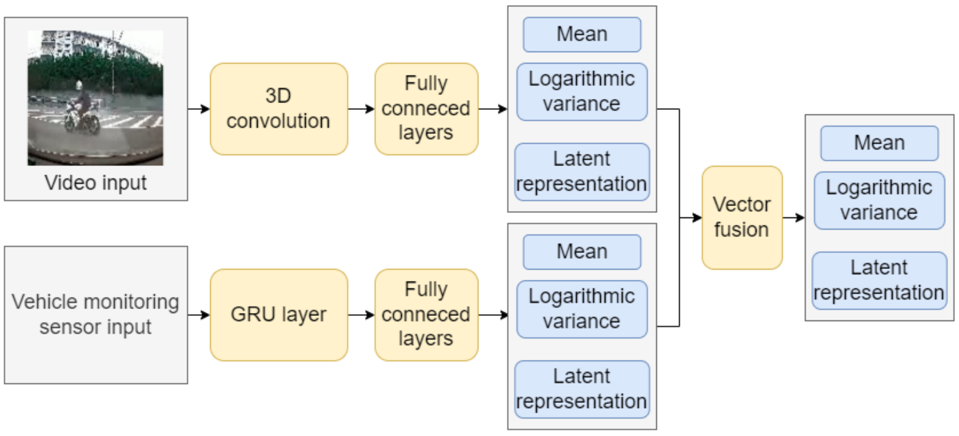


Figure 4.3: MVAE encoder structure

Again, the decoders of both models have the same structure as the encoder, only in reverse. As in the Video TVAE architecture described in subsection 4.2.1, the transformer decoder is slightly different, as it uses the unprocessed output from the transformer encoder as memory.

4.2.3 Multi-modal Autoencoders

Finally, the multi-modal AE implementations. There are three implementations:

- **Multi-Modal Semi-Variational Autoencoder (MSVAE)**
- **Multi-Modal Variational Autoencoder (MVAE)**
- **Multi-Modal Transformer Variational Autoencoder (MTVAE)**

The first two implementations are illustrated in Figure 4.3, the difference being in the vector fusion stage, and that the MSVAE model does not produce a mean and logarithmic variance of the latent representation from the video input. The MTVAE model is illustrated in Figure 4.4. All three models can be compared to their single-modality counterparts to answer RQ3.

The first implementation is a combination of the Video AE and VMS VAE, combining the two simplest individual AEs. The two modalities are encoded individually before the sampled VMS representation is concatenated with the encoded video representation at the most abstract level possible.

The second implementation combines the Video VAE and the VMS VAE. Instead of combining the encoded vectors through concatenation, this model does so by employing an attention layer, which extracts the most useful parts of both encoded modalities. These two models provide insight for answering RQ2 from section 1.2, allowing comparisons between concatenation and attention layers as means for modality fusion.

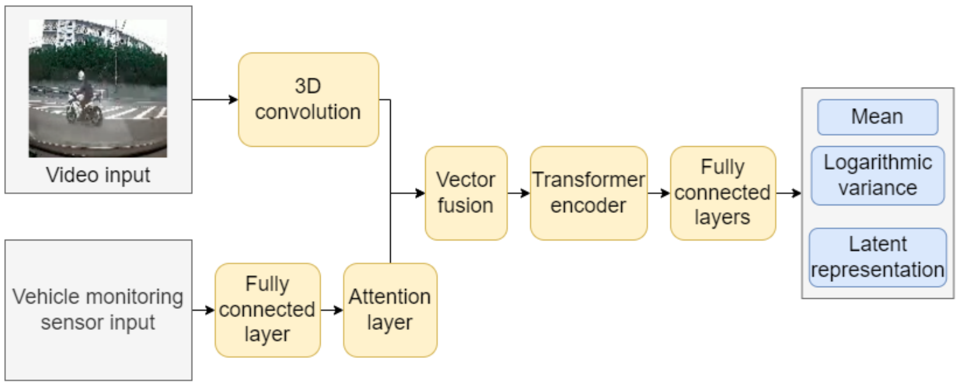


Figure 4.4: MTVAE encoder structure

The last implementation combines the Video Transformer VAE and the VMS Transformer VAE. The modalities are processed by some of the layers, to obtain the same shape, then combined by concatenation. The fused modalities are then passed through the transformer encoder layer, before a fully connected layer. Finally, two fully connected layers output the mean and logarithmic variance of the distribution of the latent representation. When compared to the MSVAE model, this model allows comparisons between high-level and mid-level modality fusion, which is of interest for RQ2.

4.2.4 Hierarchical Variational Autoencoder

The last model that is tested in these experiments, is the HVAE. The way it is implemented, it functions more as an extension of an AE. By using an already trained model, the HVAE takes the encoded vectors as input and encodes and decodes this as a regular VAE. The model then returns the encoded vector from the pre-trained model concatenated with the hierarchical features. This implementation has two hierarchical layers and utilises the MTVAE model to explore whether there is any improvement between the two. This model is tested as another possible approach for RQ1, to see if using HVAEs can be of interest for generalising the latent space.

Although this implementation works as a downstream task by using the output of a different model, one could have the hierarchical layers trained simultaneously. In this fashion, the first layer could use the concatenated output from both layers for decoding. However, for practical reasons such as training efficiency and back-propagation, the outlined implementation is chosen for the reported experiment.

4.2.5 Loss functions

The final loss computation is a combination of selected loss functions, that were previously described in sections 2.3 and 2.5.

- **MSE loss** is used for the reconstruction task, comparing the reconstructed samples to the original input. This loss function is the basis of the AE model and ensures the latent representations contain enough information to reconstruct the original data sample, thus ensuring a meaningful representation that contains extracted features from the raw input. This loss function is computed as shown in Equation 2.1.
- **KL divergence** is used to regularise the latent space and ensure that it contains more information than what is needed to reconstruct a sample. This is also used to shape the distribution latent samples are drawn from, and the loss is computed as shown in Equation 2.5. Note that the Video AE does not apply KL divergence during training.
- **Triplet loss** is applied to ensure that samples from the same driving scene are closer together in the latent space, and further away from samples from other scenes. As explained in section 4.1, this ensures to an extent that closeness in the latent space translates to a similarity in original data samples. The triplet loss that is implemented in this work is based on the Euclidean distance between a latent vector (anchor), the mean of the latent vectors from the same sequence (positive), and the closest mean of a different sequence (negative). Using these values, and a margin value of 2, the calculations are as shown in Equation 2.11.
- **L2 loss** is also used, to prevent overfitting the weights to the data in the training set and keeping the values of the weights small. This should help generalise the models and handle unseen data more accurately. It is implemented as shown in Equation 2.3, with a lambda value of 0.01.

The final loss function is presented in Equation 4.1.

$$L = MSE + KL + Triplet + L2 \tag{4.1}$$

4.3 Model Training

This section describes the various hyperparameters[2] and other specifics related to training and testing ML models. As mentioned in section 1.3, the

[2]Hyperparameter is a common term in ML practices for values that are used to control the training process.

goal of this thesis is not optimisation. Nonetheless, the results must be adequate for comparison purposes to address the research questions from section 1.2. Another reason for not extensively fine-tuning the hyperparameters is the testing of different AE implementations. Finding an optimal set of hyperparameters for each model would be too time-consuming, and fine-tuning based on the performance of only one model would lead to biased results. Consequently, the hyperparameters are primarily tuned to prevent exploding or vanishing gradients. They are selected based on the task's complexity, the size of the dataset, and brief tuning based on the reconstruction results.

Batch Size The first hyperparameter is the batch size, meaning the number of samples that are processed at the same time. Using a batch size that is too large can lead to poor generalisation, but so does using a batch size of 1. As the loss is calculated per batch, the batch size also affects how often the model weights are updated, which can also affect performance. Therefore, it is important, and not always straightforward, to find a reasonable batch size. Based on the size of the available data in this report, a **batch size of 32** is used, which is commonly chosen as a starting point [88, 89].

Epochs An epoch is the common ML term to describe the number of times to process the whole dataset. After a certain number of epochs, the test loss will stop to decrease, and might even increase if the model starts to overfit to the training data. Therefore it is important to use a sensible number of epochs or implement early stopping. In the experiments of this thesis, **the model from the epoch with the lowest test loss** is saved, which is a form of early stopping. Training for **150 epochs** allows the training for all the different models to converge.

Learning Rate (LR) The LR is the hyperparameter that determines how big the step is when updating the model weights. Typical values lay in the range of less than 1 and 10ˆ6 [90]. A **value of 0.00001** is used for the experiments, as higher LR values lead to exploding gradients, and lower values lead to slower convergence. Research has also found a correlation between batch size and LR [88], where a small batch size performs better with a smaller LR.

LR Scheduler While a certain LR value can produce good results in the early stages of training, the value might be too high in the latter stages. In the beginning, the model weights are far away from an optimal solution and thus benefit from large steps towards the optima. However, when getting closer, large steps might cause the model to overshoot and miss the optimal solution. Therefore, an LR scheduler is applied, which reduces the LR after a certain amount of time. Specifically, a scheduler that reduces the LR when the test loss has reached a plateau and does not decrease for a given number of epochs, namely **ReduceLROnPlateau** from PyTorch [87]. Although this implementation ensures that the LR does not decrease while the test loss is improving, fluctuations in the test loss might prevent the scheduler from working properly.

Optimizer An optimizer is the algorithm that updates the model weights, based on the loss values and LR. Some of the common optimizers include Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSprop), and **Adaptive Moment Estimation algorithm (Adam)** [91]. The latter is applied in the experiments of this report, with benefits including low memory requirements, and high performance for large data. Adam uses momentum when computing the updated weights, meaning that it takes into account the previous gradients of earlier increments. This allows the model to pass local optima, making it more likely to find the global optima. Adam also compares favourably to other optimizers [92] and is utilised in the experiments for these reasons.

Dropout Dropout is used to disregard certain nodes when calculating the output. The nodes are chosen at random, and the dropout rate decides how likely it is that some nodes are disregarded. This practice ensures that nodes are not codependent, and prevents overfitting. For this thesis, a **dropout rate of 0.2** is utilised in hidden layers. The value was chosen through tuning.

Activation Functions It is common practice to use an activation function after each layer in an ML model. In the reported models the Leaky ReLU activation function is mainly used for hidden layers, except for when the output should be strictly positive. Leaky ReLU allows positive values to pass and downscales negative values, and can be described as shown in the second formula in Equation 4.2. The sigmoid activation function can be expressed as shown in the last formula in Equation 4.2 and transforms the input to the range 0-1. For this reason, the sigmoid activation function is used when values should be confined to this interval, such as predicting normalised pixel values in videos.

$$\begin{aligned} ReLU &= max(0, x) \\ LeakyReLU &= max(0.01 * x, x) \\ \sigma &= \frac{1}{1 + e^x} \end{aligned} \tag{4.2}$$

Weight Initialisation Weight initialisation can have a significant impact on the training of an ML model. It is the process of setting the model weights to small, random values. Doing so can prevent vanishing and exploding gradients. The optimal way to initialise the weights depends on the type of layer and the activation function that follows. For fully connected, convolutional, and transformer layers, Kaiming initialisation is applied with a normal distribution.

Latent Vector Size The choice of latent vector size can greatly influence performance in clustering and classification tasks. Testing found that a latent vector size of **32** provides a good basis for evaluation and comparisons, while still being more computationally effective than larger latent representations. It was also found that increasing the size of the latent representation did not significantly improve any results. The MSVAE and HVAE models produce a latent vector size of 64, because of the concatenation of vectors.

Validation Due to the size of the dataset, a validation set is not used for the reported experiments. A training and testing set is used, and the available dataset is split in a 70/30 ratio for the training set and testing set respectively. As there is a class imbalance in the dataset, the dataset is first sorted by the class label, and then equally distributed among the two sets. As two of the classes denote near misses, the risk scores should also be somewhat equally distributed as a result.

Validation sets are often used if the test set is used to tune the hyperparameters, which is the case for this thesis. The original plan was to use a larger dataset for cross-validation, which is why K-fold cross-validation[3] is not utilised. However, the time constraint of this thesis did not allow the usage of a different dataset. Therefore, this part of the experiments is designated as future work and is further discussed in 6.3.

4.4 Evaluation

This section describes the process of evaluating the produced latent representations. Three different tasks are chosen for this purpose:

- **Reconstruction**
- **Clustering**
- **Classification**

[3]K-fold cross-validation evaluates a model's performance for unseen data by splitting the training set into K folds

Figure 4.5 illustrates the evaluation process. As this thesis mainly focuses on how well the latent spaces generalise to different downstream tasks, the illustration only includes the clustering and classification tasks. The research questions given in section 1.2 are addressed by evaluating these tasks.

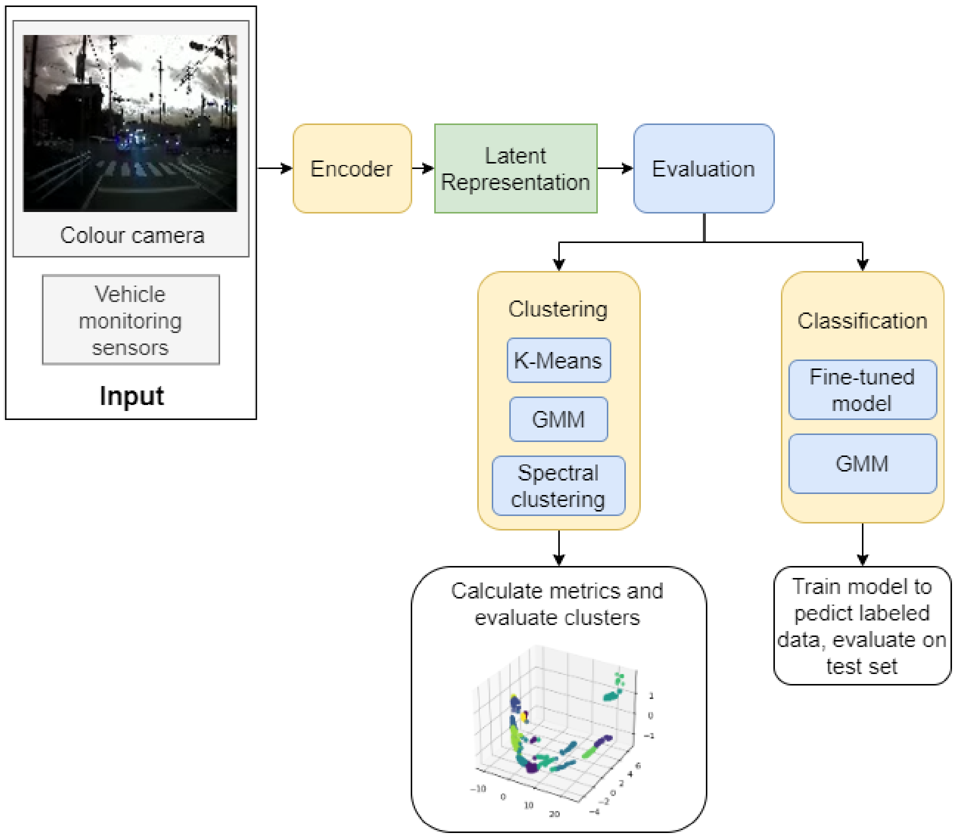


Figure 4.5: Flowchart of model inference

4.4.1 Reconstruction

No additional implementation is needed for this task, as it simply evaluates the performance of the training objective of the model. To evaluate this task, the test set is used to calculate the Mean Square Error between the original and reconstructed samples. Additionally, the Mean Absolute Percentage Error (MAPE) score is provided to give a more intuitive metric that describes the mean error in percentages. While the best possible score is 0, the upper limit is not restricted to 1. It is important to note that when the true values are close to 0, the MAPE score can give arbitrarily high values, which should be taken into consideration.

4.4.2 Clustering

Three different methods are utilised for the clustering task to examine whether there is an underlying pattern in the latent space that works best with a specific method. The three forms of clustering applied in the reported experiments are:

- **K-Means**
- **Spectral Clustering**
- **Gaussian Mixture Models**

The first step in the evaluation process is to produce latent representations for the full test set, by using the encoders of the AE implementations. Using this set of latent representations, the three clustering algorithms are applied with increasing K values. To evaluate the resulting clusters, both the Silhouette score and Calinski-Harabasz are calculated and demonstrate how well the driving scenes are separated in the latent space. As described in subsection 4.1, each driving scene is divided into 4 separate data samples and used for calculating the triplet loss outlined in subsection 2.5.1. Thus, the resulting latent space should reflect similarity in the real-world driving scenes.

All three cluster algorithms and cluster evaluation methods are implemented using the scikit-learn library [93] and are outlined in subsection 2.5.1.

4.4.3 Classification

The objective of this task is to classify two aspects: class labels that indicate the type of driving scenario, and RiskScores, which measure the risk of a driving scene. The purpose is to assess the amount of information that can be obtained from the latent representation. The clustering task determines to which degree the driving scenes are distinguishable in the latent space, while this task evaluates the adaptability of the latent space to particular tasks.

Class labels To assign class labels to the driving scenes based on their latent representations, two methods were utilised: fine-tuned model and GMM. The fine-tuned model, which was explained in subsection 2.5.2, involves training a small ML model with two fully connected layers to classify the samples. The model utilises the ReLU activation function and outputs the probability for each class. CE loss is used to train the model, with weights accounting for the class imbalance of the dataset. The latter approach combines GMMs with logistic regression. The fine-tuned model was implemented using PyTorch [87], while the sci-kit [93] library was used for the GMM method. Even though the fine-tuned model has the advantage of requiring less labelled data, these experiments were conducted with the same training and testing data as the reconstruction task to ensure adequate results. Testing the fine-tuned model with less data would be part of any future work, and is mentioned in section 6.3.

Risk score In this task, a fine-tuned model is utilised to assign a risk score to each driving scene. As the risk scores are continuous values, the model architecture for this differs slightly from the one used for label classification. The model generates a single value, which is passed through a modified sigmoid activation function to limit the output between -5 and 5. To train the model, MSE loss is implemented.

The class distribution depicted in Figure 3.2 reveals that there are only 276 samples from classes 13 and 14, which include near misses. This suggests an imbalance in the distribution of risk scores as well, which can affect the ability of the model to accurately classify high-risk driving scenes.

Classification task metrics Accuracy is a common metric for classification tasks and measures how often a label is correctly predicted. However, for data sets with uneven class distributions, this metric does not adequately measure performance. Therefore, precision and recall are often used. The former measures the proportion of true positives among all predicted positives, while the latter measures the proportion of true positives among all actual positives. Another option is the F1 score, which is the harmonic mean of precision and recall. A form of visualisation that is often used, is the confusion matrix, which shows the number of correctly and incorrectly predicted samples for each class. The equations for each of the four metrics are shown in the list of equations 4.3.

$$\begin{aligned}
\text{Accuracy} &= \frac{TP + TN}{TP + TN + FP + FN} \\
\text{Precision} &= \frac{TP}{TP + FP} \\
\text{Recall} &= \frac{TP}{TP + FN} \\
\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} &= \frac{2 \cdot TP}{2 \cdot TP + FP + FN}
\end{aligned} \tag{4.3}$$

For continuous values, metrics such as MAPE and MAE can be used to measure how close the predictions are on average.

Chapter 5

Experiments

The chapter consists of five sections. The first, **section 5.1**, describes the results of the reconstruction task. **Section 5.2** presents and analyses the results of the clustering task. **Section 5.3** describes the results of the classification task, using both a fine-tuned model and a GMM classifier. The last one, **section 5.4**, summarises the results. By comparing the AE variations in these tasks, the chapter addresses the research questions from section 1.2.

5.1 Reconstruction task

In this section, the results of the reconstruction task are presented and analysed. Although the primary focus of the research is not on the reconstruction task, it is still important to assess how well the different implementations can reconstruct the input data since the task is used to train the models. Moreover, it allows for the examination of whether there is a correlation between performance in this task and clustering or classification, which is not always guaranteed. At the very least, the ability to decode an encoded vector to reconstruct the original data sample demonstrates that the latent representation contains extracted features.

For this task, the MSE between reconstructed and original data samples is used for evaluation. However, as MSE does not provide a particularly intuitive value for how close the predicted values are, MAPE is also included as a metric. The MAPE values are given such that a score of 100 is equivalent to 100%. Because of the way MAPE is calculated, it can output very large values when the true values are close to zero. As this is the case for the VMS data, where values are often zero, both the true and predicted values of the VMS data are incremented by 1 to get a better estimate. These values are shown for each model in Table 5.1.

To visualise the difference between the reconstructed and original Video data, the first frame from a reconstructed video is included in Figure 5.1. For comparison, the original frame is shown in Figure 5.2.

Table 5.1: MSE and MAPE for reconstruction task

Model	**Video MSE**	**VMS MSE**	**Video MAPE**	**VMS MAPE**
Video AE	328824	N/A	**2.439**	N/A
Video VAE	**321625**	N/A	2.486	N/A
Video TVAE	354947	N/A	2.443	N/A
VMS VAE	N/A	**4516**	N/A	**1.030**
VMS TVAE	N/A	46738	N/A	1.461
MSVAE	341024	10403	2.616	1.124
MVAE	326049	47362	2.524	1.212
MTVAE	581364	57970	4.152	1.467

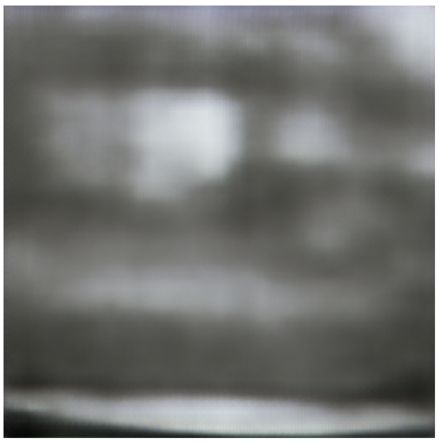


Figure 5.1: Reconstructed frame by Video VAE

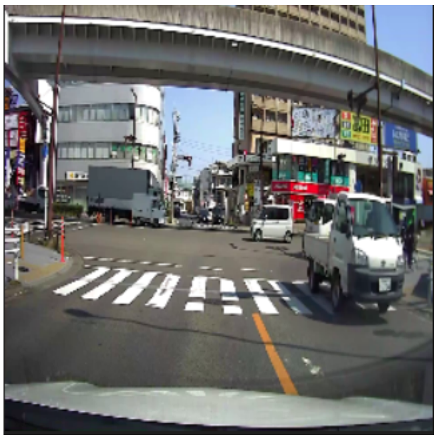


Figure 5.2: Frame from original video

5.1.1 Analysis of Reconstruction Results

Judging by the MAPE values in Table 5.1, the reconstructed samples are quite close to the original ones. The results demonstrate that all models are capable of creating latent representations that can be decoded to reproduce the input data. The models perform quite similarly, however, the VMS VAE and the MTVAE stand out. The first model is the only one to significantly outperform the others, with a MAPE score of **1.03%**. The latter performs considerably worse in reconstructing the video data. A possible explanation for this is that the VMS TVAE performs poorly compared to the GRU-based implementation, and thus affects the reconstruction of video data when combined in the MTVAE architecture.

Although the MAPE scores present a promising mean difference in the range of 1-4%, Figures 5.1 and 5.2 visualise this difference between the original and reconstructed video input better. The reconstructed image fails to capture details and colours from the original input. It includes coarse differences in lighting, but identifying objects in the reconstructed image is quite difficult.

The reconstruction results of the HVAE model are not included, as the model reconstructs a given latent representation. Thus, it does not make sense to compare this to the other models.

5.2 Clustering

In this section, the latent spaces are assessed using three clustering algorithms: K-Means, spectral clustering, and GMM clustering. However, since all three methods yield similar results, this section focuses on the K-Means method. The results obtained from the other two algorithms are included in the appendix. The compactness and separation of the clusters are evaluated using the silhouette and CH scores. Different K values ranging from 2 to 14 are tested to determine the number of clusters that the data samples can be grouped into. Up to 14 clusters are tested due to the 14 categories in the class labels. Due to the CH score not having an upper bound, there is a large variance in the scores. The CH plot in Figure 5.3 is therefore presented with a logarithmic y-axis. The silhouette score is presented in Figure 5.4. To visualise the clusters, PCA has been applied to present the latent space in a 3D plot, as shown in Figures 5.5 and 5.6. The two figures illustrate the difference between the highest- and lowest-scoring models in both the silhouette and CH scores.

As mentioned in subsection 2.5.1, K-Means is a hard algorithm, meaning that repeating the experiment would yield the same results every time.

5.2.1 Analysis of Clustering Results

As explained in subsection 2.5.1 for the latter metric, 1 is the best value, which indicates that the clusters are well separated. Values near 0 indicate overlapping clusters. As a baseline for the clustering task, a silhouette score of 0 can be used, as this would be the same as randomly assigning samples to clusters. Also explained in subsection 2.5.1, the CH score measures the variance of the clusters, and a higher score indicates more compact clusters.

All the models perform better than the baseline with 2 clusters, except the HVAE model, which performs significantly worse than the other models in both metrics. As the number of clusters increases, most of the models only perform slightly better than the baseline. The VMS VAE, Video TVAE, and the VMS TVAE outperform the other models for two clusters, with over **0.6** in silhouette score. As K increases, the VMS VAE and MTVAE perform well, with a silhouette score of **0.34** for K = 4. The performance decreases drastically as K increases past 2 clusters, which proves to be the highest number of well-separated clusters.

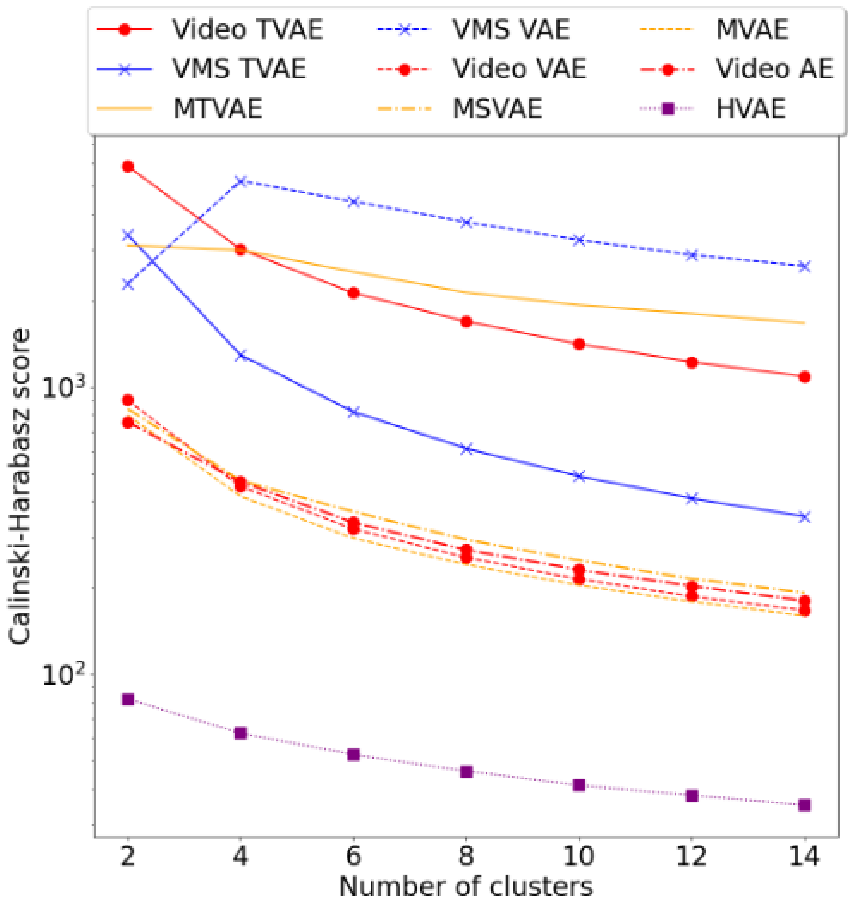


Figure 5.3: Calinski-Harabasz scores for K-Means clusters

The axis values shown in Figures 5.5 and 5.6 demonstrate the difference in the size of the latent spaces. Although this is the PCA reduced and not actual latent spaces, it shows that the VMS VAE produces latent representations that are more spread out.

5.3 Classification Task

This section describes experiments for classifying both class labels and risk scores, using fine-tuned models and GMMs. The experiments are done to examine how descriptive the latent representations are, and what type of information can be extracted from it. As classifying the original 14 labels can be a demanding task, the experiment is done using only three classes, each with the same number of classes. Classification results for all 14 classes

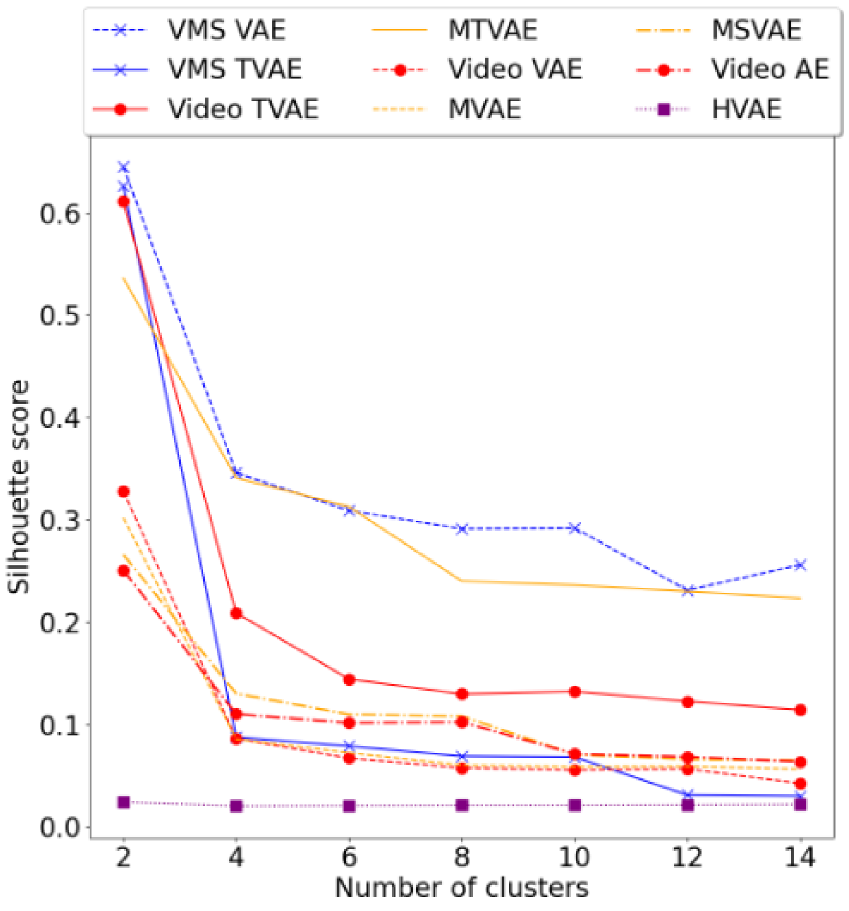


Figure 5.4: Silhouette scores for K-Means clusters

can be found in the appendix in Table 7.1.

It is acceptable to use accuracy as an evaluation metric since the three chosen classes have an equal number of samples. As all classes have equal probability, random guessing would lead to an accuracy of 0.33, which is used as a baseline. The results of both the fine-tuned and GMM label classification are given in Table 5.2. The confusion matrix of the highest-scoring model is shown in Figure 5.7.

For the risk score task, the MAPE score is used to evaluate each model. Due to the actual risk scores being close to zero, the MAPE scores are quite high. Therefore, the Mean Absolute Error is also included. Assigning a value of zero for every driving scene results in a MAPE score of 100% and MAE of 0.367. These values are used as a baseline, as this is the strategy that provides the best score without making informed predictions. The results are presented in Table 5.3.

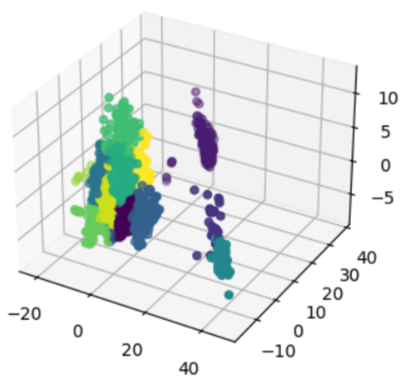


Figure 5.5: VMS VAE clusters after PCA, with K = 14

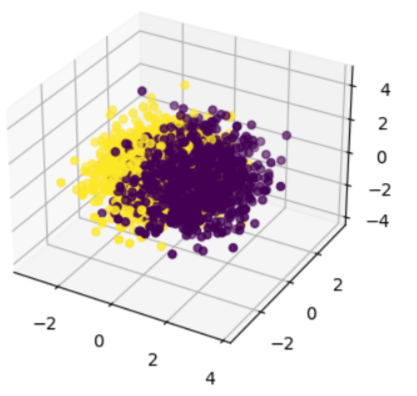


Figure 5.6: HVAE clusters after PCA, with K = 2

Table 5.2: Fine-tuned model and GMM classification results for 3 classes

Model	**Fine-tuned**		**GMM**	
	F1	**Accuracy**	**F1**	**Accuracy**
Video AE	0.332	0.369	0.292	0.300
Video VAE	0.220	0.34	0.334	0.342
Video TVAE	0.295	0.360	0.337	0.34
VMS VAE	0.340	0.343	0.166	0.293
VMS TVAE	0.325	0.328	0.175	0.311
MSVAE	0.345	0.351	0.309	0.341
MVAE	**0.382**	**0.382**	0.387	0.391
MTVAE	0.300	0.312	0.208	0.261
HVAE	0.298	0.301	**0.388**	**0.400**

5.3.1 Analysis of Classification Results

This subsection analyses and highlights the important results from the classification experiments.

GMM label classification The two models that achieve the highest scores are the **MVAE and HVAE**, with **39.1% and 40% accuracy** respectively, which is higher than the baseline. The other models produce accuracies around or below the baseline.

Repeating the GMM classification 100 times with the MVAE model demonstrates a standard deviation of 0.05 for accuracy and 0.06 for the F1 score. Thus, assuming all models have a similar standard deviation, the only models that can be said to outperform the baseline with confidence is the MVAE and MTVAE models.

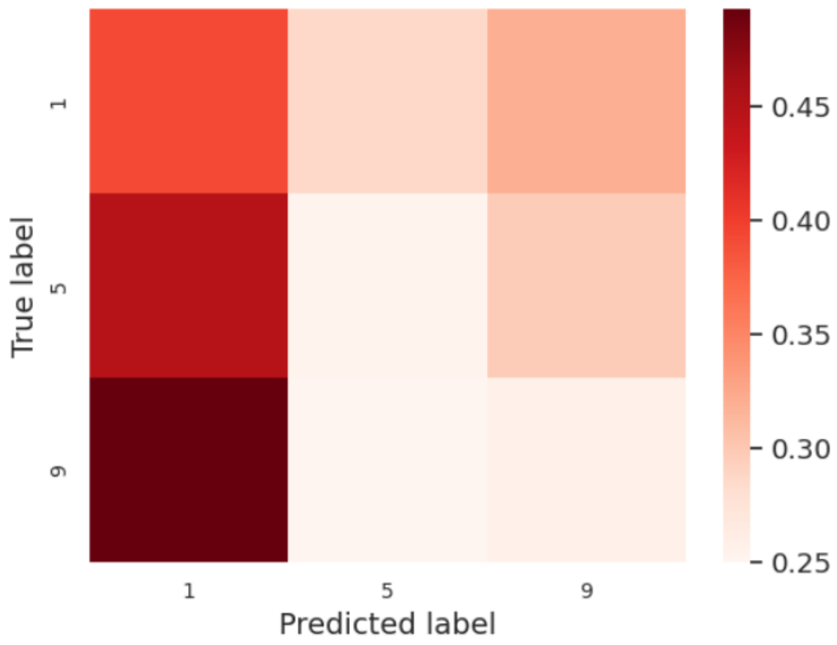


Figure 5.7: Confusion matrix for HVAE with GMM classification

Fine-tuned label classification The **MVAE** achieves the highest score in this experiment, with an **accuracy of 38.2%**. Again, most of the models achieve an accuracy close to or below the baseline.

Because the training time of the fine-tuned models is significantly longer than the GMMs, the experiment is only repeated 10 times, resulting in a standard deviation of 0.023 and 0.029 for the accuracy and F1 score respectively. The standard deviation was calculated using the MVAE model. As sample sizes increase, standard deviations usually decrease, indicating that the standard deviation of this task could be smaller. The results of this experiment can then be considered more reliable than the results of the GMM classification.

Fine-tuned risk score classification In the last classification task, assigning risk scores using a fine-tuned model, the **VMS TVAE and Video AE** achieve the highest scores, with **0.383 and 0.413 MAE** respectively. All models produce quite large errors percentage-wise, as shown by the MAPE scores in Table 5.3, none of which outperform the baseline. Due to this, this experiment will not be considered as much as the others in the final evaluation.

The poor performance in this task could stem from the utilised activation

Table 5.3: Risk prediction scores for fine-tuned models

Model	MAPE	MAE
Video AE	151	0.413
Video VAE	571	0.539
Video TVAE	1950	1.277
VMS VAE	6245	4.562
VMS TVAE	**147**	**0.383**
MSVAE	7560	5.013
MVAE	765	0.660
MTVAE	300	0.439
HVAE	405	0.495

function. The range of -5 to 5 of the modified sigmoid function might make it difficult to accurately predict values close to 0, where a large portion of the samples is located. Although there is some variance in the results of this performance, no model consistently performs better than the baseline. Because of the large variance in performance between different models, a single measure of standard deviation does not accurately portray the variance in each model.

5.4 Discussion

In this section the research questions from section 1.2 are discussed. Although the results can be improved upon, they provide a satisfactory basis for comparisons. The models perform better than the baseline in all tasks, except for risk classification.

Research Question 1 The first research question asks how features can be extracted from each modality using self-supervised methods. This thesis examines the use of autoencoders for this task and compares the performance of regular and variational AEs. The results are similar for the Video AE and Video VAE across all tasks, making it difficult to draw any conclusions about this. The usage of hierarchical VAEs is also studied, which showed promising results in the label classification tasks, especially in predicting class labels using GMMs. However, the clustering results of the HVAE are exceptionally poor. Given the results in other tasks, this could indicate an error in the clustering process. Both of the AE variations warrant further testing in any future work.

The three Video Autoencoder implementations perform similarly in the reconstruction and label classification tasks. In the clustering tasks, the Video TVAE outperforms the other two, while for risk classification, it underperforms compared to the other two. Given the generally poor results in the risk classification task, it suggests that the transformer approach is worth exploring further.

The two VMS Autoencoder variations achieve similar scores in the classification tasks. However, in the reconstruction and clustering tasks, the GRU-based implementation significantly outperforms the other model, indicating that using a transformer layer for the VMS data is not beneficial.

Research Question 2 The second research question asks how modalities can be combined to enhance the performance of the single-modality components. As stated in section 4.2, the three approaches to modality fusion are high-level concatenation, high-level attention layer, and mid-level concatenation.

The first approach, high-level concatenation, performs slightly worse than its components in the reconstruction task. For the clustering and fine-tuned label classification tasks, the results are similar, and thus no significant improvement. In the GMM classification task, the results have improved. However, accounting for the high standard deviation of this task, the improvement is not significant.

The second approach, which utilises the high-level attention layer, performs well in both fine-tuned and GMM-based driving scenario classification, compared to the single modality AEs. However, there is no significant improvement in the clustering task.

The last approach, mid-level concatenation, is the only fusion approach that significantly increases the clustering results. Although not as notable as the clustering task, the results for classifying risk scores also improve utilising this method. In the other tasks, this approach does not increase the performance.

Research Question 3 Lastly, the third research question asks whether performance improves when modalities are combined. It depends on both the fusion method and task, as each experiment provides various answers. The most significant improvement can be seen in the clustering task for the mid-level concatenation. The attention layer approach also improves results in the classification tasks.

Chapter 6

Future Work

In this chapter future work is described. As the results of the experiments described in chapter 5 indicate, improvements should be made to the model implementations. Therefore, any future work should primarily involve such improvements, as described in **section 6.1**. Given promising results from this, adding other modalities to the framework is also of interest, as **section 6.2** explains. **Section 6.3** discusses the need for a validation set. Finally, **section 6.4** describes potential applications that are of interest for future work.

6.1 Model improvements

The primary goal of any future work would be to improve the models for the tasks already tested in this thesis. There are always improvements to be made, and while optimisation is not the main objective of this thesis, there are some improvements that could be worth testing in future research. Any improvement of individual or multi-modal models would be a step towards the original goal: producing a useful and generalised latent space using SSL. It would also be easier to further evaluate the three research questions from section 1.2 if performance improves.

Given the summary in section 5.4, some variants should be tested in future work. As the GRU layer performed better than the transformer layer for VMS data, a MTVAE variant with GRU layers should be tested. Similarly, the modality fusion with attention layers demonstrated promising results. Therefore, a model that utilises mid-level fusion with attention layers should be evaluated.

6.1.1 Improvement of VMS Variational Autoencoder

After further research and understanding of the NEDO dataset and the type of data it contains, there are some modifications to the VMS VAE model that are interesting. It seems that there are data points in some feature sequences that are missing. Currently, this is not explicitly handled. Those missing data points are filled using interpolation in the data preparation stage, which might not be the optimal solution. The Variational Autoencoder might allow better reconstruction of the missing data samples, due to the method of sampling a probable representation from a distribution. Thus, these missing data samples would be masked in the data preparation stage, rather than filled by interpolation or similar techniques.

In addition, it seems that many of the categorical variables represent states that do not change throughout the driving scene. Other features represent states that could change over time, but since the driving scene only lasts around seven seconds, they seldom do change. The static features could be separated and encoded in a different model in future work.

For increased performance, certain features could be highlighted. Although it is preferred to avoid human bias in selecting and handling features, some features such as the velocity and acceleration of the ego vehicle could be processed differently. As Zhu et al. [83] do in their paper, the velocity and acceleration can be Fourier[1] transformed to obtain characterisations of driver behaviour. This would be an interesting approach for any future work, as it would allow the transformation of a 1-dimensional feature sequence to a 2-dimensional vector. This can be layered on top of the video data for low-level modality fusion. Any other approaches that enable low-level fusion are also of interest for future work.

6.1.2 Loss Function

As explained in subsection 4.2.5, triplet loss is included as part of the loss function used to train the models. This is done by splitting a driving scene into four different sequences and using the mean Euclidian distance of these to calculate the triplet loss. The intended effect of this is to ensure to some degree that closeness in the latent space is translated to a similarity in the driving scenes. However, the loss function can be improved in future work by testing different measures of distance. Another improvement could be to calculate the triplet loss with individual sequences, instead of the mean of sequences from each driving scene. In addition to these improvements, weighting the different components of the loss function would also be an aspect worth testing. It is worth revisiting the loss function in any future work, as it is the basis of shaping the latent space and the information it contains.

[1]Fourier transform is the mathematical process of transferring a signal from one domain to another.

6.1.3 Additional Self-Supervised Task

Another option for future work would be a structural change to the training framework. By training an AE to encode and decode individual frames and timesteps, the resulting latent representation would be a sequence of extracted features. Another model, such as the CPC model briefly described in section 2.4, can then be used to learn a representation of this sequence. Other interesting approaches for future work include testing other forms of sequence compression and compare to the current approach. As described in section 4.2, the current approach is a simple one: flattening the sequential vector and applying fully connected layers. Thus, this stage of the model training has the potential for improvement in any future work.

6.1.4 Ensemble Learning

As the AEs are trained on the augmented dataset, the models expect the sequence length of the inputs to be a certain length. For this reason, it is difficult to make a single prediction for a whole driving scene. However, a possible remedy for this is ensemble learning, which is the process of having multiple models make predictions, and combining the outputs to enhance accuracy. Sagi and Rokach [94] review different methods of ensemble learning. Taking inspiration from this, future work can combine the classification for each sequence of a driving scene, and study whether this improves performance.

6.2 Add Modalities

This thesis uses input from a colour camera sensor, in addition to a collection of vehicle sensors. In future work, adding other modalities would be interesting to research. As described in section 2.2, depth sensors are commonly used in autonomous driving. Thus, after an adequate improvement to the already implemented models, future work with a different dataset that contains depth sensor outputs would be of interest.

6.3 Validation

Given adequate improvements in current tasks, future work should test the framework on a larger dataset. This would allow the creation and usage of the planned validation set, as mentioned in section 4.3. Future work should also compare the proposed methods from this thesis with supervised methods, by training a classifier with similar architecture as the encoder from each model on labelled data. The classifiers and fine-tuned models should be trained on a subset of the larger dataset. This would allow the evaluation of whether the fine-tuned models perform better than supervised methods when labelled data is limited, which is one of the main benefits of fine-tuning, as discussed in section 1.1. Another benefit of utilising a larger dataset is that the size of the models can be increased. The size of the current dataset limits the optimal size of a model, but the complexity of the task of extracting features from video and VMS data might require a larger model.

6.4 Applications

If the performance is improved in the reported tasks, other applications could be worth exploring in future work.

6.4.1 Clustering applications

As mentioned in section 1.1, anomaly detection is a common application of clusters. To do this, there must be anomalies in the dataset. Depending on which type of anomaly is desirable to detect, there are several methods to do this. If the goal is to detect corrupted data, this can be added to the dataset by adding noise to individual samples, or occluding parts of the input data.

Another interesting path for future work is the real-time clustering of new driving scenes. Given a previously collected dataset, which is divided into clusters, live driving scenes can easily be grouped with similar scenes. This would be a form of recognising and identifying what kind of driving scenario the ego vehicle is in.

6.4.2 Classification Tasks

In this thesis, GMMs and fine-tuned models are used for classification tasks. Other classification methods might perform better, such as random forest classification [95] or the Support Vector Machine (SVM) model, which performs well with non-linear data. Another approach to improve the classification experiments could be to perform feature selection of the latent representation. There is a possibility that not all latent features are needed for certain tasks, such as classifying the risk of a scene. Approaches of interest include recursive feature elimination and random forest importance.

The NEDO dataset also contains action tags, environment tags, and scene descriptions. If model improvements lead to reasonable performances in the fine-tuning tasks that are tested in this work, other fine-tuning tasks can be tested in future research as well. Training a model to provide scene descriptions would be a highly useful application, as it would be a step towards explainable AI. This could augment the results of clustering and other potential tasks by justifying decisions made by the model. Explainable AI is the research field of making decisions made by AI models more transparent, and therefore more trustworthy [96]. This is especially important in sectors such as healthcare, finance, and autonomous driving.

Chapter 7

Conclusion

This thesis proposes methods that use self-supervised learning methods to extract features from video and Vehicle Monitoring Sensor input. The objective is to represent the extracted features as latent representations, which can be used for clustering and classification tasks. This method is applied using a dataset containing real-world driving scenes. The research consists of training several Machine Learning model variations, based on the Autoencoder structure. Although state-of-the-art performance is not achieved in reconstructing the original video and vehicle monitoring sensor data, the models perform adequately in this task. This demonstrates that the models are trained correctly and can produce valid latent representations.

Finally, the model variations are evaluated by using produced latent representations for clustering and classification tasks. The Vehicle Monitoring Sensor VAE and the Multi-Modal Transformer Variational Autoencoder models achieve a silhouette score of 0.34 for 4 clusters with the K-Means algorithm. The best result for the classification tasks is achieved by the Multi-Modal Variational Autoencoder model, with an accuracy of 38.2% for 3 classes. These results reveal that the implementations presented in this thesis are not sufficient for use in classification tasks, and should be improved upon in future work. However, they also demonstrate that the approach is better than random guessing, thus indicating that adjustments can lead to enhanced performance. The clustering results show that the Vehicle Monitoring Sensor VAE and Multi-Modal Transformer Variational Autoencoder can produce latent representations that are sufficiently distinct to be separated into four clusters.

In conclusion, the methods that provide the best results for research question 1 are using transformers for video data, and GRU layers for VMS data. For research question 2, the best approach proved to be mid-level concatenation. However, mid-level fusion with attention layers could further enhance the performance and should be tested in future work. Finally, for research question 3, the experiments reveal that mid-level fusion significantly improves results in the clustering task.

Appendix

Hardware

The reported experiments were conducted on a machine with these specifications:

- **CPU:** AMD EPYC 7F72 24-Core
- **GPU:** NVIDIA RTX A6000

Software

These are the software specifications used for the reported experiments:

- **OS** Ubuntu 20.04.5 LTS
- **Python** 3.8.10
- **torch** 1.13.1
- **torchvision** 0.14.1
- **numpy** 1.24.2
- **matplotlib** 3.7.0
- **pandas** 1.5.3
- **pickle-mixin** 1.0.2
- **seaborn** 0.12.2
- **Pillow** 9.4.0

Code

The code used for the experiments of this thesis is available in this GitHub repository:
https://github.com/andrseassundfjord/thesis/tree/main

Additional Results

This section includes additional results that are not significant to the main objectives of this thesis.

Table 7.1: Fine-tuned model and GMM classification results for 14 classes

Model	Fine-tuned		GMM	
	F1	**Accuracy**	**F1**	**Accuracy**
Video AE	0.0654	0.0877	0.0582	0.0722
Video VAE	0.0805	0.0864	**0.0866**	**0.1145**
Video TVAE	0.0508	0.0833	0.0636	0.0765
VMS VAE	0.0744	0.0793	0.0403	0.0633
VMS TVAE	0.0779	0.0873	0.0761	**0.1145**
MSVAE	0.0766	0.0815	0.0326	0.0383
MVAE	**0.0895**	**0.0969**	0.0592	0.0873
MTVAE	0.0833	0.0920	0.0663	0.0941
HVAE	0.0757	0.0852	0.0585	0.0685

Table 7.2: MAPE for reconstructed masked features in VMS data. NaN values indicate a very large difference or a programming error.

Model	**MAPE**
VMS VAE	**1.774**
VMS TVAE	1.853
MSVAE	NaN
MVAE	NaN
MTVAE	NaN

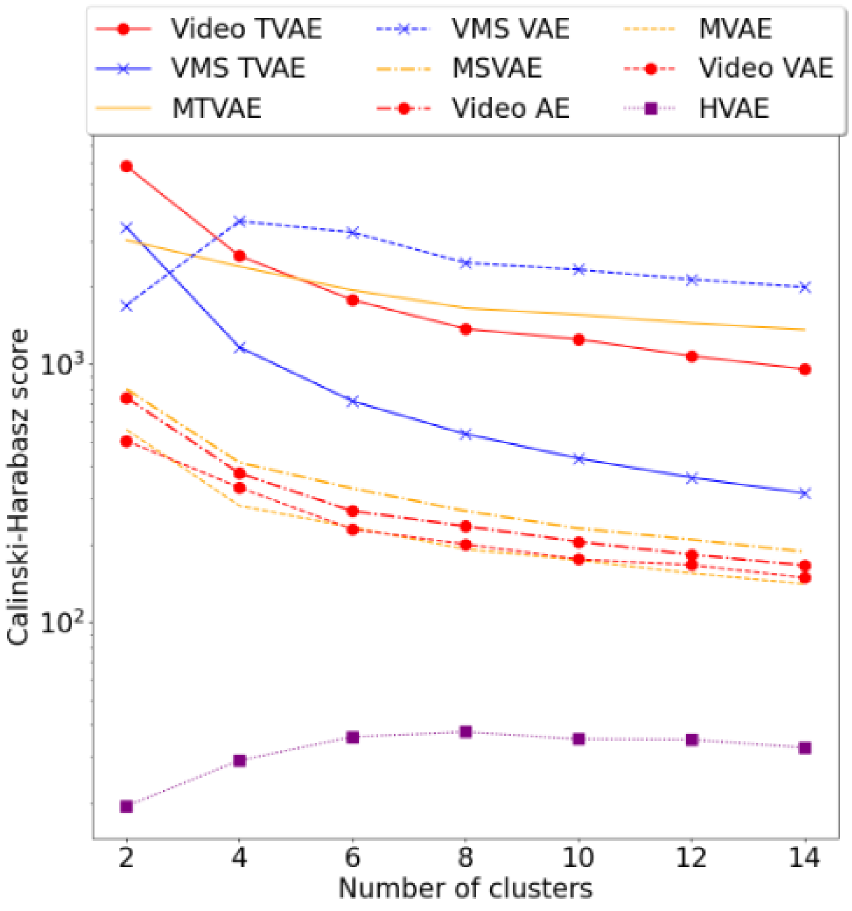


Figure 7.1: Calinski-Harabasz scores for GMM clusters

Table 7.3: ARI and V Measure scores for K-Means cluster prediction of class labels

Model	**ARI**	**V Measure**
Video AE	0.02611	0.07883
Video VAE	0.02996	0.08739
Video TVAE	0.01553	0.05726
VMS VAE	**0.04926**	**0.13371**
VMS TVAE	0.00181	0.01922
MSVAE	0.02718	0.07785
MVAE	0.02666	0.08183
MTVAE	0.01457	0.05499
HVAE	-0.00014	0.01039

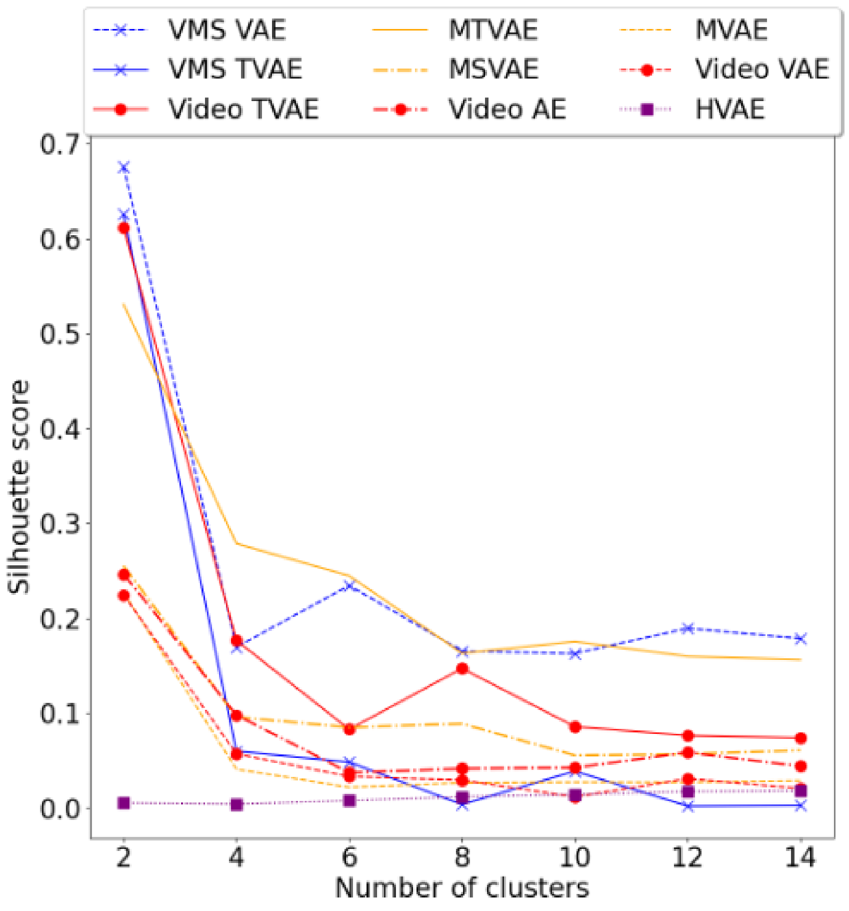


Figure 7.2: Silhouette scores for GMM clusters

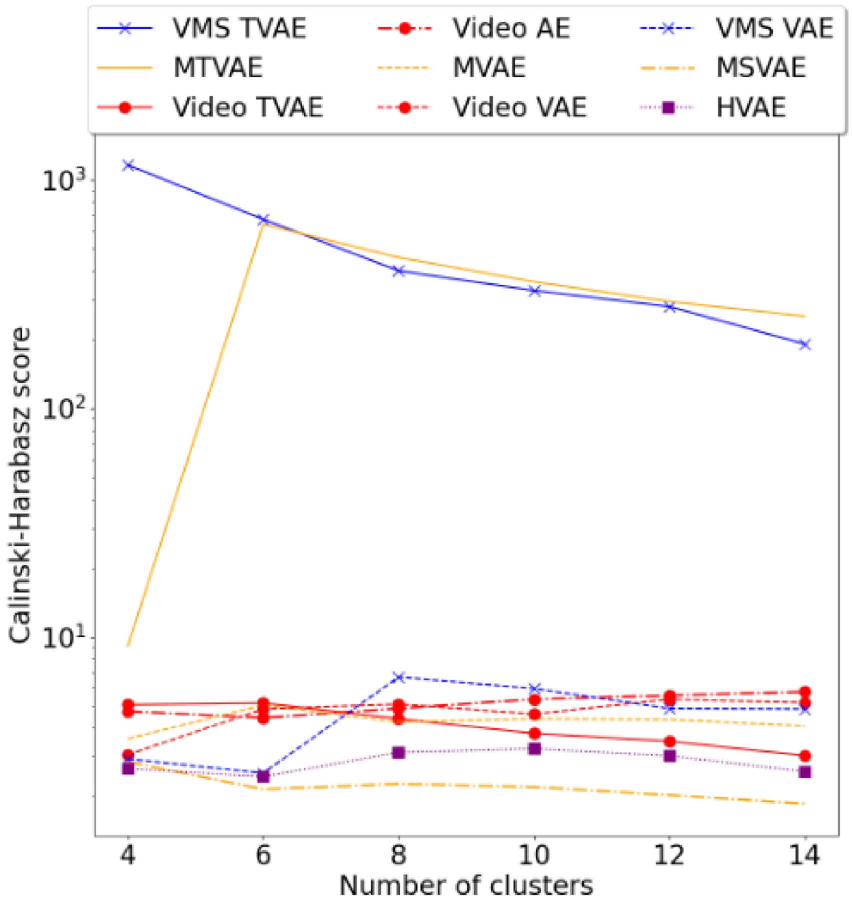


Figure 7.3: Calinski-Harabasz scores for spectral clusters

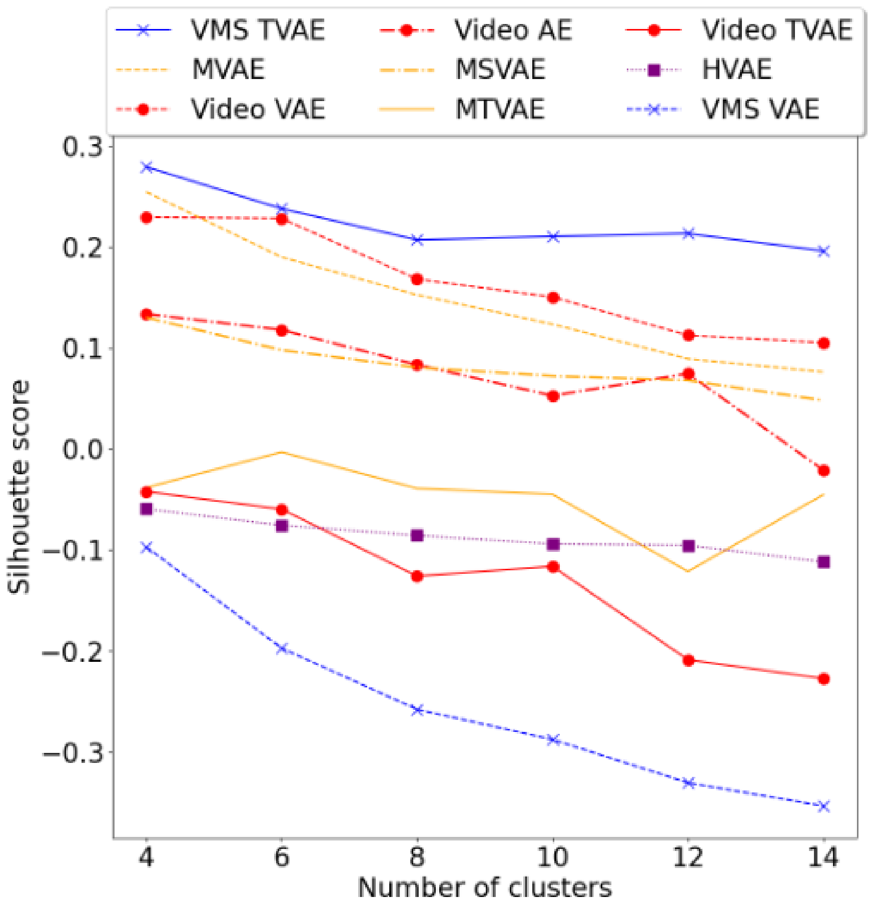


Figure 7.4: Silhouette scores for spectral clusters

Bibliography

[1] Mohd. Hafiz Hasan and Pascal Van Hentenryck. 'The benefits of autonomous vehicles for community-based trip sharing'. In: *Transportation Research Part C: Emerging Technologies* 124 (2021), p. 102929. ISSN: 0968-090X. DOI: https://doi.org/10.1016/j.trc.2020.102929. URL: https://www.sciencedirect.com/science/article/pii/S0968090X20308287.

[2] W David Montgomery. 'Public and private benefits of autonomous vehicles'. In: (2018).

[3] Simone Pettigrew, Zenobia Talati and Richard Norman. 'The health benefits of autonomous vehicles: Public awareness and receptivity in Australia'. In: *Australian and New Zealand journal of public health* 42.5 (2018), pp. 480–483.

[4] John R Treat. 'A study of precrash factors involved in traffic accidents.' In: *HSRI Research review* (1980).

[5] John R Treat, Nicholas S Tumbas, Stephen T McDonald, David Shinar, Rex D Hume, RE Mayer, RL Stansifer and N John Castellan. *Tri-level study of the causes of traffic accidents: final report. Executive summary.* Tech. rep. Indiana University, Bloomington, Institute for Research in Public Safety, 1979.

[6] Fridulv Sagberg. 'Road accidents caused by drivers falling asleep'. In: *Accident Analysis & Prevention* 31.6 (1999), pp. 639–649. ISSN: 0001-4575. DOI: https://doi.org/10.1016/S0001-4575(99)00023-8. URL: https://www.sciencedirect.com/science/article/pii/S0001457599000238.

[7] Jeff K. Caird, Kate A. Johnston, Chelsea R. Willness, Mark Asbridge and Piers Steel. 'A meta-analysis of the effects of texting on driving'. In: *Accident Analysis & Prevention* 71 (2014), pp. 311–318. ISSN: 0001-4575. DOI: https://doi.org/10.1016/j.aap.2014.06.005. URL: https://www.sciencedirect.com/science/article/pii/S000145751400178X.

[8] Saeed Asadi Bagloee, Madjid Tavana, Mohsen Asadi and Tracey Oliver. 'Autonomous vehicles: challenges, opportunities, and future implications for transportation policies'. In: *Journal of modern transportation* 24 (2016), pp. 284–303.

[9] Jeremy Howard and Sebastian Ruder. *Universal Language Model Fine-tuning for Text Classification.* 2018. arXiv: 1801.06146 [cs.CL].

[10] Ryan Florin and Stephan Olariu. 'Towards real-time density estimation using vehicle-to-vehicle communications'. In: *Transportation research part B: methodological* 138 (2020), pp. 435–456.

[11] Jian-Ru Xue, Jian-Wu Fang and Pu Zhang. 'A survey of scene understanding by event reasoning in autonomous driving'. In: *International Journal of Automation and Computing* 15.3 (2018), pp. 249–266.

[12] Vasili Ramanishka, Yi-Ting Chen, Teruhisa Misu and Kate Saenko. *Toward Driving Scene Understanding: A Dataset for Learning Driver Behavior and Causal Reasoning.* 2018. arXiv: 1811.02307 [cs.CV].

[13] Hongkuan Zhang, Koichi Takeda, Ryohei Sasano, Yusuke Adachi and Kento Ohtani. 'Driving behavior aware caption generation for egocentric driving videos using in-vehicle sensors'. In: *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops).* IEEE. 2021, pp. 287–292.

[14] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck and Klaus Dietmayer. 'Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges'. In: *IEEE Transactions on Intelligent Transportation Systems* 22.3 (2020), pp. 1341–1360.

[15] Esraa Khatab, Ahmed Onsy, Martin Varley and Ahmed Abouelfarag. 'Vulnerable objects detection for autonomous driving: A review'. In: *Integration* 78 (2021), pp. 36–48.

[16] Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S Ecker, Matthias Bethge and Wieland Brendel. 'Benchmarking robustness in object detection: Autonomous driving when winter is coming'. In: *arXiv preprint arXiv:1907.07484* (2019).

[17] Yingjie Niu, Ming Ding, Yuxiao Zhang, Kento Ohtani and Kazuya Takeda. 'Auditory and visual warning information generation of the risk object in driving scenes based on weakly supervised learning'. In: *2022 IEEE Intelligent Vehicles Symposium (IV).* 2022, pp. 1572–1577. DOI: 10.1109/IV51971.2022.9827382.

[18] Jessica Shea Choksey and Christian Wardlaw. *Levels of Autonomous Driving, Explained.* https://www.jdpower.com/cars/shopping-guides/levels-of-autonomous-driving-explained. Accessed: 2023-05-28.

[19] Pranav Dixit. *Mercedes beats Tesla in self-driving, becomes first certified Level-3 autonomous car company in US.* https://www.businesstoday.in/technology/news/story/mercedes-beats-tesla-in-self-driving-becomes-first-certified-level-3-autonomous-car-company-in-us-367937-2023-01-28. Accessed: 2023-05-28.

[20] Gonzalo De La Torre, Paul Rad and Kim-Kwang Raymond Choo. 'Driverless vehicle security: Challenges and future research opportunities'. In: *Future Generation Computer Systems* 108 (2020), pp. 1092–1111. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2017.12.041. URL: https://www.sciencedirect.com/science/article/pii/S0167739X17315066.

[21] Margarita Martínez-Díaz and Francesc Soriguera. 'Autonomous vehicles: theoretical and practical challenges'. In: *Transportation Research Procedia* 33 (2018). XIII Conference on Transport Engineering, CIT2018, pp. 275–282. ISSN: 2352-1465. DOI: https://doi.org/10.1016/j.trpro.2018.10.103. URL: https://www.sciencedirect.com/science/article/pii/S2352146518302606.

[22] Luca Caltagirone, Mauro Bellone, Lennart Svensson and Mattias Wahde. 'LIDAR–camera fusion for road detection using fully convolutional neural networks'. In: *Robotics and Autonomous Systems* 111 (2019), pp. 125–131.

[23] Feihu Zhang, Daniel Clarke and Alois Knoll. 'Vehicle detection based on LiDAR and camera fusion'. In: *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2014, pp. 1620–1625.

[24] Pål Primstad. 'A Thermal and RGB feature extraction system for use in autonomous navigation'. MA thesis. 2020.

[25] Dudi Nassi, Raz Ben-Netanel, Yuval Elovici and Ben Nassi. *MobilBye: Attacking ADAS with Camera Spoofing.* 2019. arXiv: 1906.09765 [cs.CR].

[26] Farzan Majeed Noori, Michael Riegler, Md Zia Uddin and Jim Torresen. 'Human Activity Recognition from Multiple Sensors Data Using Multi-Fusion Representations and CNNs'. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 16.2 (May 2020). ISSN: 1551-6857. DOI: 10.1145/3377882. URL: https://doi.org/10.1145/3377882.

[27] David Bordvik, Jie Hou, Farzan Majeed Noori, Md. Zia Uddin and Jim Torresen. 'Monitoring In-Home Emergency Situation and Preserve Privacy using Multi-modal Sensing and Deep Learning'. In: Feb. 2022, pp. 1–6. DOI: 10.1109/ICEIC54506.2022.9748829.

[28] De Jong Yeong, Gustavo Velasco-Hernandez, John Barry and Joseph Walsh. 'Sensor and Sensor Fusion Technology in Autonomous Vehicles: A Review'. In: *Sensors* 21.6 (2021). ISSN: 1424-8220. DOI: 10.3390/s21062140. URL: https://www.mdpi.com/1424-8220/21/6/2140.

[29] Mani Kumar Tellamekala, Shahin Amiriparian, Björn W. Schuller, Elisabeth André, Timo Giesbrecht and Michel Valstar. *COLD Fusion: Calibrated and Ordinal Latent Distribution Fusion for Uncertainty-Aware Multimodal Emotion Recognition.* 2022. arXiv: 2206.05833 [cs.CV].

[30] Jelena Kocić, Nenad Jovičić and Vujo Drndarević. 'Sensors and Sensor Fusion in Autonomous Vehicles'. In: *2018 26th Telecommunications Forum (TELFOR).* 2018, pp. 420–425. DOI: 10.1109/TELFOR.2018.8612054.

[31] Giulia Rizzoli, Francesco Barbato and Pietro Zanuttigh. 'Multimodal Semantic Segmentation in Autonomous Driving: A Review of Current Approaches and Future Perspectives'. In: *Technologies* 10.4 (2022). ISSN: 2227-7080. DOI: 10.3390/technologies10040090. URL: https://www.mdpi.com/2227-7080/10/4/90.

[32] Chen Fu, Christoph Mertz and John M Dolan. 'Lidar and monocular camera fusion: On-road depth completion for autonomous driving'. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC).* IEEE. 2019, pp. 273–278.

[33] Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao and Deyi Li. 'Object classification using CNN-based fusion of vision and LIDAR in autonomous vehicle environment'. In: *IEEE Transactions on Industrial Informatics* 14.9 (2018), pp. 4224–4231.

[34] Kinjal Dasgupta, Arindam Das, Sudip Das, Ujjwal Bhattacharya and Senthil Yogamani. 'Spatio-Contextual Deep Network-Based Multimodal Pedestrian Detection for Autonomous Driving'. In: *IEEE Transactions on Intelligent Transportation Systems* 23.9 (2022), pp. 15940–15950. DOI: 10.1109/TITS.2022.3146575.

[35] Ravi Yadav, Ahmed Samir, Hazem Rashed, Senthil Yogamani and Rozenn Dahyot. 'Cnn based color and thermal image fusion for object detection in automated driving'. In: *Irish Machine Vision and Image Processing* (2020).

[36] Zhiyu Huang, Chen Lv, Yang Xing and Jingda Wu. 'Multi-Modal Sensor Fusion-Based Deep Neural Network for End-to-End Autonomous Driving With Scene Understanding'. In: *IEEE Sensors Journal* 21.10 (May 2021), pp. 11781–11790. DOI: 10.1109/jsen.2020.3003121. URL: https://doi.org/10.1109%5C%2Fjsen.2020.3003121.

[37] Arc. *Convolutional Neural Network.* https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05. Accessed: 2023-06-05.

[38] Paul-Louis Pröve. *An Introduction to different Types of Convolutions in Deep Learning.* https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d. Accessed: 2023-06-05.

[39] Giuliano Giacaglia. *How Transformers Work.* https://towardsdatascience.com/transformers-141e32e69591. Accessed: 2023-06-05.

[40] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk and Yoshua Bengio. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.* 2014. arXiv: 1406.1078 [cs.CL].

[41] Sepp Hochreiter and Jürgen Schmidhuber. 'Long short-term memory'. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[42] Shudong Yang, Xueying Yu and Ying Zhou. 'LSTM and GRU Neural Network Performance Comparison Study: Taking Yelp Review Dataset as an Example'. In: *2020 International Workshop on Electronic Communication and Artificial Intelligence (IWECAI).* 2020, pp. 98–101. DOI: 10.1109/IWECAI50956.2020.00027.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser and Illia Polosukhin. 'Attention is All you Need'. In: *Advances in Neural Information Processing Systems.* Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[44] Shigeki Karita, Nanxin Chen, Tomoki Hayashi, Takaaki Hori, Hirofumi Inaguma, Ziyan Jiang, Masao Someki, Nelson Enrique Yalta Soplin, Ryuichi Yamamoto, Xiaofei Wang et al. 'A comparative study on transformer vs rnn in speech applications'. In: *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU).* IEEE. 2019, pp. 449–456.

[45] Rui Wang, Dongdong Chen, Zuxuan Wu, Yinpeng Chen, Xiyang Dai, Mengchen Liu, Yu-Gang Jiang, Luowei Zhou and Lu Yuan. 'BEVT: BERT Pretraining of Video Transformers'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* June 2022, pp. 14733–14743.

[46] Javier Selva, Anders S. Johansen, Sergio Escalera, Kamal Nasrollahi, Thomas B. Moeslund and Albert Clapés. 'Video Transformers: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023), pp. 1–20. DOI: 10.1109/TPAMI.2023.3243465.

[47] Shamane Siriwardhana, Tharindu Kaluarachchi, Mark Billinghurst and Suranga Nanayakkara. 'Multimodal Emotion Recognition With Transformer-Based Self Supervised Feature Fusion'. In: *IEEE Access* 8 (2020), pp. 176274–176285. DOI: 10.1109/ACCESS.2020.3026823.

[48] Ransaka Ravihara. *Gaussian Mixture Model Clearly Explained.* http://jalammar.github.io/illustrated-transformer/. Accessed: 2023-06-05.

[49] Seunghyoung Ryu, Hyungeun Choi, Hyoseop Lee and Hongseok Kim. 'Convolutional Autoencoder Based Feature Extraction and Clustering for Customer Load Analysis'. In: *IEEE Transactions on Power Systems* 35.2 (2020), pp. 1048–1060. DOI: 10.1109/TPWRS.2019.2936293.

[50] Yongshan Zhang, Yang Wang, Xiaohong Chen, Xinwei Jiang and Yicong Zhou. 'Spectral–Spatial Feature Extraction With Dual Graph Autoencoder for Hyperspectral Image Clustering'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 32.12 (2022), pp. 8500–8511. DOI: 10.1109/TCSVT.2022.3196679.

[51] Chunyong Yin, Sun Zhang, Jin Wang and Neal N. Xiong. 'Anomaly Detection Based on Convolutional Recurrent Autoencoder for IoT Time Series'. In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.1 (2022), pp. 112–122. DOI: 10.1109/TSMC.2020.2968516.

[52] Aaron van den Oord, Yazhe Li and Oriol Vinyals. 'Representation learning with contrastive predictive coding'. In: *arXiv preprint arXiv:1807.03748* (2018).

[53] Limin Wang, Bingkun Huang, Zhiyu Zhao, Zhan Tong, Yinan He, Yi Wang, Yali Wang and Yu Qiao. *VideoMAE V2: Scaling Video Masked Autoencoders with Dual Masking.* 2023. arXiv: 2303.16727 [cs.CV].

[54] Arden Dertat. *Applied Deep Learning - Part 3: Autoencoders.* https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798. Accessed: 2023-06-05.

[55] QuanLin Wu, Hang Ye, Yuntian Gu, Huishuai Zhang, Liwei Wang and Di He. 'Denoising Masked Autoencoders Help Robust Classification'. In: *The Eleventh International Conference on Learning Representations.* 2023. URL: https://openreview.net/forum?id=zDjtZZBZtqK.

[56] Jinxin Zhao, Jin Fang, Zhixian Ye and Liangjun Zhang. *Large Scale Autonomous Driving Scenarios Clustering with Self-supervised Feature Extraction.* 2021. arXiv: 2103.16101 [cs.CV].

[57] Wangyang Wei, Honghai Wu and Huadong Ma. 'An AutoEncoder and LSTM-Based Traffic Flow Prediction Method'. In: *Sensors* 19.13 (2019). ISSN: 1424-8220. DOI: 10.3390/s19132946. URL: https://www.mdpi.com/1424-8220/19/13/2946.

[58] Abhyuday Desai, Cynthia Freeman, Zuhui Wang and Ian Beaver. *TimeVAE: A Variational Auto-Encoder for Multivariate Time Series Generation.* 2021. arXiv: 2111.08095 [cs.LG].

[59] Xiulan Yu, Hongyu Li, Zufan Zhang and Chenquan Gan. 'The Optimally Designed Variational Autoencoder Networks for Clustering and Recovery of Incomplete Multimedia Data'. In: *Sensors* 19.4 (2019). ISSN: 1424-8220. DOI: 10.3390/s19040809. URL: https://www.mdpi.com/1424-8220/19/4/809.

[60] Robin Karlsson, Alexander Carballo, Keisuke Fujii, Kento Ohtani and Kazuya Takeda. *Predictive World Models from Real-World Partial Observations.* 2023. arXiv: 2301.04783 [cs.CV].

[61] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee and Andrew Y Ng. 'Multimodal deep learning'. In: *Proceedings of the 28th international conference on machine learning (ICML-11).* 2011, pp. 689–696.

[62] Dhruv Khattar, Jaipal Singh Goud, Manish Gupta and Vasudeva Varma. 'Mvae: Multimodal variational autoencoder for fake news detection'. In: *The world wide web conference.* 2019, pp. 2915–2921.

[63] Kyle Sama, Yoichi Morales, Hailong Liu, Naoki Akai, Alexander Carballo, Eijiro Takeuchi and Kazuya Takeda. 'Extracting human-like driving behaviors from expert driver data using deep learning'. In: *IEEE transactions on vehicular technology* 69.9 (2020), pp. 9315–9329.

[64] Eva Patel and Dharmender Singh Kushwaha. 'Clustering Cloud Workloads: K-Means vs Gaussian Mixture Model'. In: *Procedia Computer Science* 171 (2020). Third International Conference on Computing and Network Communications (CoCoNet'19), pp. 158–167. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.04.017. URL: https://www.sciencedirect.com/science/article/pii/S1877050920309820.

[65] Paul D. McNicholas. 'Model-based classification using latent Gaussian mixture models'. In: *Journal of Statistical Planning and Inference* 140.5 (2010), pp. 1175–1181. ISSN: 0378-3758. DOI: https://doi.org/10.1016/j.jspi.2009.11.006. URL: https://www.sciencedirect.com/science/article/pii/S0378375809003607.

[66] Jay Alammar. *The Illustrated Transformer.* https://towardsdatascience.com/gaussian-mixture-model-clearly-explained-115010f7d4cf. Accessed: 2023-06-05.

[67] S. Charles Brubaker. *Proceedings of the 2009 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA).* Ed. by Claire Mathieu. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2009. DOI: 10.1137/1.9781611973068. eprint: https://epubs.siam.org/doi/pdf/

10.1137/1.9781611973068. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611973068.

[68] R. Hadsell, S. Chopra and Y. LeCun. 'Dimensionality Reduction by Learning an Invariant Mapping'. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. 2006, pp. 1735–1742. DOI: 10.1109/CVPR.2006.100.

[69] Florian Schroff, Dmitry Kalenichenko and James Philbin. 'FaceNet: A unified embedding for face recognition and clustering'. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298682. URL: https://doi.org/10.1109%5C%2Fcvpr.2015.7298682.

[70] Xingping Dong and Jianbing Shen. 'Triplet Loss in Siamese Network for Object Tracking'. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[71] Sahil Sharma and Vijay Kumar. '3D landmark-based face restoration for recognition using variational autoencoder and triplet loss'. In: *IET Biometrics* 10.1 (2021), pp. 87–98.

[72] Kaiwei Zeng, Munan Ning, Yaohua Wang and Yang Guo. 'Hierarchical clustering with hard-batch triplet loss for person re-identification'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 13657–13665.

[73] Yuanjie Yan, Hongyan Hao, Baile Xu, Jian Zhao and Furao Shen. 'Image clustering via deep embedded dimensionality reduction and probability-based triplet loss'. In: *IEEE Transactions on Image Processing* 29 (2020), pp. 5652–5661.

[74] Şaban Öztürk and Tolga Çukur. 'Deep clustering via center-oriented margin free-triplet loss for skin lesion detection in highly imbalanced datasets'. In: *IEEE Journal of Biomedical and Health Informatics* 26.9 (2022), pp. 4679–4690.

[75] Jinbo Li, Hesam Izakian, Witold Pedrycz and Iqbal Jamal. 'Clustering-based anomaly detection in multivariate time series data'. In: *Applied Soft Computing* 100 (2021), p. 106919. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2020.106919. URL: https://www.sciencedirect.com/science/article/pii/S1568494620308577.

[76] Yuqi Ouyang and Victor Sanchez. 'Video Anomaly Detection by Estimating Likelihood of Representations'. In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021, pp. 8984–8991. DOI: 10.1109/ICPR48806.2021.9412694.

[77] Neelum Noreen, Sellapan Palaniappan, Abdul Qayyum, Iftikhar Ahmad and Madini O Alassafi. 'Brain Tumor Classification Based on Fine-Tuned Models and the Ensemble Method.' In: *Computers, Materials & Continua* 67.3 (2021).

[78] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: 1810.04805 [cs.CL].

[79] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong and Qing He. 'A Comprehensive Survey on Transfer Learning'. In: *Proceedings of the IEEE* 109.1 (2021), pp. 43–76. DOI: 10.1109/JPROC.2020.3004555.

[80] Rensis Likert, Sydney Roslow and Gardner Murphy. 'A Simple and Reliable Method of Scoring the Thurstone Attitude Scales'. In: *The Journal of Social Psychology* 5.2 (1934), pp. 228–238. DOI: 10.1080/00224545.1934.9919450. eprint: https://doi.org/10.1080/00224545.1934.9919450. URL: https://doi.org/10.1080/00224545.1934.9919450.

[81] Jing Wang, ZhongCheng Wu, Fang Li and Jun Zhang. 'A Data Augmentation Approach to Distracted Driving Detection'. In: *Future Internet* 13.1 (2021). ISSN: 1999-5903. DOI: 10.3390/fi13010001. URL: https://www.mdpi.com/1999-5903/13/1/1.

[82] Sangdoo Yun, Seong Joon Oh, Byeongho Heo, Dongyoon Han and Jinhyung Kim. *VideoMix: Rethinking Data Augmentation for Video Classification.* 2020. arXiv: 2012.03457 [cs.CV].

[83] Shengxue Zhu, Chongyi Li, Kexin Fang, Yichuan Peng, Yuming Jiang and Yajie Zou. 'An Optimized Algorithm for Dangerous Driving Behavior Identification Based on Unbalanced Data'. In: *Electronics* 11.10 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11101557. URL: https://www.mdpi.com/2079-9292/11/10/1557.

[84] Mwamba Kasongo Dahouda and Inwhee Joe. 'A deep-learned embedding technique for categorical features encoding'. In: *IEEE Access* 9 (2021), pp. 114381–114391.

[85] Cheng Guo and Felix Berkhahn. 'Entity embeddings of categorical variables'. In: *arXiv preprint arXiv:1604.06737* (2016).

[86] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern recognition.* Elsevier, 2006.

[87] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai and Soumith Chintala. *PyTorch: An Im-*

perative Style, High-Performance Deep Learning Library. 2019. arXiv: 1912.01703 [cs.LG].

[88] Ibrahem Kandel and Mauro Castelli. 'The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset'. In: *ICT Express* 6.4 (2020), pp. 312–315. ISSN: 2405-9595. DOI: https://doi.org/10.1016/j.icte.2020.04.010. URL: https://www.sciencedirect.com/science/article/pii/S2405959519303455.

[89] Yoshua Bengio. 'Practical recommendations for gradient-based training of deep architectures'. In: *Neural Networks: Tricks of the Trade: Second Edition* (2012), pp. 437–478.

[90] Yoshua Bengio. 'Practical recommendations for gradient-based training of deep architectures'. In: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533. URL: http://arxiv.org/abs/1206.5533.

[91] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[92] Robin M. Schmidt, Frank Schneider and Philipp Hennig. *Descending through a Crowded Valley - Benchmarking Deep Learning Optimizers*. 2021. arXiv: 2007.01547 [cs.LG].

[93] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay. 'Scikit-learn: Machine Learning in Python'. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[94] Omer Sagi and Lior Rokach. 'Ensemble learning: A survey'. In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 8.4 (2018), e1249.

[95] Friedrich Kruber, Jonas Wurst, Eduardo Sánchez Morales, Samarjit Chakraborty and Michael Botsch. 'Unsupervised and Supervised Learning with the Random Forest Algorithm for Traffic Scenario Clustering and Classification'. In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019, pp. 2463–2470. DOI: 10.1109/IVS.2019.8813994.

[96] Filip Karlo Došilović, Mario Brčić and Nikica Hlupić. 'Explainable artificial intelligence: A survey'. In: *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2018, pp. 0210–0215. DOI: 10.23919/MIPRO.2018.8400040.