POLITECNICO DI TORINO

Politecnico di Torino

Master degree course in Data Science and Engineering

Master Degree Thesis

Enhancing Structural Health Monitoring through Self-supervised learning: An application of Masked Autoencoders on Anomaly Detection and Traffic Load Estimation

Supervisors
Doc. Daniele Jahier Pagliari
Doc. Alessio Burrello
Doc. Luca Zanatta

Candidates
Yhorman A Bedoya V
Student number: 287296

Academic Year 2022-2023

Summary

Structural Health Monitoring (SHM) is an increasingly important field due to the rising number and complexity of structures such as bridges, buildings, and viaducts. The need to improve user safety and ensure the optimal functionality of the structures has motivated the improvement of SHM systems from sporadic human evaluations to continuous monitoring through various types of sensors capable of streaming high-frequency data

The increasing amount of SHM data has encouraged the development of algorithms and methodologies that support structural monitoring and decision-making processes in order to extend the structure's lifespan, reduce maintenance costs, and ensure the user's safety.

Anomaly detection and Traffic Load Estimation (TLE) are two key activities in the field of SHM. Anomaly detection strategies are used to identify when a structure is not functioning as expected, indicating damage in the early stages, optimizing maintenance activities, and increasing the reliability of the structures. TLE is useful to estimate the usage level of a structure at a given time. An effective TLE pipeline helps to understand patterns over time about the load of the structure, which can be useful to identify possible overload situations or to improve maintenance activities schedules.

The aim of this thesis is to study the use of Self-Supervised Transformer-Based Masked Autoencoders on data generated by SHM systems to address the tasks of anomaly detection and TLE. Additionally, we aim to demonstrate that a fine-tuning phase on a pre-trained model leads to better performance on supervised tasks than training the supervised models from scratch.

Our methodology is divided into two steps for each task. First, a general pre-training phase is common for both applications, in which the signal generated by SHM accelerometers is transformed into PSD spectrograms, The masked autoencoder is then used in a self-supervised reconstruction task to learn latent representations and patterns from the spectrograms. In the second step, the pre-trained autoencoder is used on the specific applications.

For anomaly detection, we propose a methodology based on the reconstruction error to identify anomalies. For TLE, we propose a fine-tuning phase in which the pre-trained autoencoder is further trained using a smaller labeled dataset to predict the number of vehicles crossing the viaduct at a given time based on the signal disturbances.

The results of our experiments demonstrate that we achieve an accuracy of 98.8% in the anomaly detection tasks with a 1-minute delay in detecting the anomalies. This result is comparable to the state-of-art techniques based on the PCA algorithm trained on the same dataset, which achieves an accuracy of 98.8%, but with a 1-hour delay in detecting anomalies. For TLE, we achieve a Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) of 0.4/26%. We also demonstrate that a pre-training phase improves the model's performance by 20%.

Keywords: Structural Health Monitoring, Masked Autoencoders, Transformers, Traffic Load Estimation, Anomaly Detection

Acknowledgements

Quisiera dedicar un agradecimiento a mis padres. Este logro es el fruto de su esfuerzo, dedicación y sacrificio durante todos estos años. A mi padre, Octavio, quien siempre nos ha mostrado que el trabajo duro y la dedicación son el mejor camino para alcanzar cualquier objetivo. Le agradezco por trabajar incansablemente cada día para dar a su familia una vida digna y tranquila. A mi madre, Patricia, quien ha dedicado su vida a cuidar de su familia. Le agradezco porque sin su constante apoyo, incluso desde la distancia, no habría sido posible alcanzar este objetivo.

Agradezco a mis supervisores Daniele Jahier Pagliari, Alessio Burrello y Luca Zanatta por su ayuda, disponibilidad y todo lo que aprendí de ustedes durante el desarrollo de esta tesis. Fue un placer haber trabajado bajo su supervisión.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Structural Health Monitoring (SHM) is the implementation of damage identification strategies for aerospace, civil, and mechanical engineering infrastructures. According to [13], SHM activities aim to detect damage in structures early, monitor their condition continuously, prevent catastrophic events, extend their lifespan, save maintenance costs, and ensure user safety.

For many years, SHM activities relied on human observation and evaluation. However, these methods have several drawbacks, including human subjectivity, human inability to detect small changes, high cost, and risk to human safety when the evaluated structures are deteriorated by the lack of maintenance or after extraordinary situations such as natural disasters. To overcome these issues, it is crucial to develop automatic SHM methodologies that can continuously and efficiently monitor the state of the structures.

During the past few years, Internet of Things (IoT) systems have been implemented to support Structural Health Monitoring (SHM) activities. IoT devices are primarily utilized to gather and transmit real-time data related to various properties of the structure, such as vibration, temperature, and humidity. These systems enable real-time monitoring of the structures and offer other advantages, including high accuracy in detecting even small changes in the structure's behavior, minimized monitoring costs, and scalability, which allows monitoring of different structures from a single location.

To develop efficient methodologies that support SHM activities using data from IoT sensors, it is necessary to create algorithms capable of transforming the generated data into useful information to facilitate the decision-making process. The most common SHM applications using data from IoT systems include anomaly detection algorithms, damage classifications, usage level estimations, or time series analyses on usage patterns. Through our research,

we have found that most of the automatic SHM applications rely on classical machine learning algorithms, such as Principal Component Analysis (PCA) [29], support vector machines [6], and random forests [6], among others. However, methodologies based on deep learning architectures have not been significantly studied for SHM applications.

Deep learning is a subfield of Machine Learning that takes inspiration from the structure and functioning of the human brain. These models rely on neural networks composed of multiple layers of interconnected nodes, where input data flows hierarchically to extract abstract features beneficial for solving diverse kinds of problems like classifications or regression. Deep learning models offer several advantages over classical ML models. For example, DL models can learn more efficiently on large amounts of data, extracting more complex patterns and generating better performances. They can also handle data in its raw form, while ML models require a manual feature extraction process to perform well. Additionally, Deep learning models are more flexible and adaptable to various tasks and data types. For instance, architectures like transformers have been utilized in computer vision and natural language processing (NLP) fields, achieving state-of-the-art results.

This work is focused on the application of deep learning algorithms for SHM tasks. We propose a methodology based on self-supervised learning to handle anomaly detection and TLE tasks on two distinct viaducts located in Italy. These viaducts have been continuously monitored by a vibrational-based SHM sensor network that uses vibration-based measurements to detect damages and track the viaducts' health condition.

The acceleration data obtained from the sensor networks is segmented into overlapping windows, filtered, and transformed into Power Spectral Density (PSD) spectrograms. These spectrograms are employed to pre-train a Masked Autoencoder (MAE) through a self-supervised reconstruction task. During this pre-training phase, the model learns the general dependencies and patterns present in the data, which can later be applied to specific SHM applications.

In the anomaly detection task, the dataset contains both normal and anomaly data. The model is pre-trained on the reconstruction task exclusively on normal data, and a statistical threshold is used on the reconstruction error to identify anomaly data. In the case of TLE, the dataset comprises a vast collection of vibration data without any labels and a smaller set of vibration data coupled with the corresponding vehicle information generated by a scale as they pass over the viaduct. The model is pre-trained with the non-labeled data to perform reconstruction tasks. Then, the pre-trained model's

architecture is altered, and it is further trained with the labeled dataset to estimate the number of vehicles crossing the viaduct based on the vibration data.

The proposed methodologies are compared with the state-of-the-art applications presented on [29] and [6]. As result, the proposed anomaly detection pipeline reaches an accuracy of 99.4%, with a sensitivity and specificity of 99.8% and 98.65% respectively, this result is evidence of the capability of the proposed methodology to accurately detect anomalies, outperforming the state-of-the-art application and

To evaluate our proposed methodologies, we compared our results with state-of-the-art applications. On anomaly detection, The state-of-the-art methodology presented in [29] reports an accuracy of 98.8%, with a sensitivity and specificity of 97.33% and 100%, respectively, for detecting an anomaly with a detection delay of 60 minutes. In contrast, our anomaly detection pipeline achieves an accuracy of 99.4%, with a sensitivity and specificity of 99.8% and 98.65%, respectively, on aggregation windows of 100 vehicles. We also explored aggregations on the time domain and achieved an accuracy of 98.8%, with a sensitivity and specificity of 99.7% and 97.4%, respectively, with a detection delay of 1 minute in the detection of the anomaly. These results demonstrate that our proposed methodology can accurately detect anomalies, outperforming the state-of-the-art method not only in detection rates but also in detection times.

To ensure a fair comparison with the state-of-the-art methodology [6] on TLE, we replicated their methodology using our data to generate a baseline representing the performance of their method on our application. The best state-of-the-art algorithm on our data was based on Random Forest (RF), which achieved an MSE and MAPE of 0.661 and 49.47%, respectively. In contrast, our proposed methodology achieved an MSE and MAPE of 0.343 and 25.58%, respectively, demonstrating a significant improvement over the state-of-the-art baseline. Additionally, we demonstrated that pre-training the regression model using self-supervised learning can improve its performance by 20% compared to a model trained directly on labeled data.

Chapter 2

Background

In this section, all the Machine Learning (ML) algorithms and techniques used during the thesis development will be introduced, as well as an introduction to the thesis application field, Structural Health Monitoring (SHM).

2.1 Machine learning and Deep learning

According to [34], Machine learning (ML) refers to the computational process through which a computer system acquires the ability to perform a specific task without being explicitly programmed. This is achieved by utilizing a learning algorithm that is fed with input data representing prior experiences. The goal of this process is to produce a computer program able to successfully perform a given task by leveraging the knowledge acquired through the input data. ML has been applied in various fields generating state-of-the-art results on tasks that are too complex for conventional algorithms or require adaptation to dynamic changes in the environment. The most common ML algorithms are for example Decision Trees, Random Forests, Support Vector Machines, regressions, K-Nearest Neighbors, and Neural Networks, among others. ML algorithms generally perform better in scenarios where the datasets are small and the features are well defined, or the computational resources are limited. However, when it comes to handling complex tasks that require higher levels of abstraction, conventional algorithms may fall short. In this case, Deep Learning algorithms normally perform better.

Deep learning is a subset of machine learning that uses artificial neural networks to model and solve complex problems. Typical deep learning algorithms are more effective than machine learning ones on unstructured data,

large datasets, and complex and nonlinear problems. Deep learning has important applications in areas such as object detection, text classification, and anomaly detection. For example, YOLOv4 [4] introduces an object detection model that achieves state-of-the-art accuracy results on different object detection benchmarks while being able to operate sufficiently fast for being used in real-time applications with a low amount of computational resources. BERT [9] introduces a pre-trained model based on Transformers that is able to capture contextual information in different texts thanks to two main features during training: the use of a bidirectional pre-training which allows the model to learn from both left and right context of a text, and the "Masked Language Model (MLM)" pre-training objective, where random tokens from an input text are masked and the model's objective is to predict the original masked tokens. This pre-training strategy allows BERT to be fine-tuned on different Natural Language Processing (NLP) tasks, resulting in state-of-the-art results on eleven of them.

2.1.1 Types of learning

ML can be divided into different learning paradigms that aim to describe the relationship between the learner algorithm and the environment, one of the most important distinctions is made by dividing the ML task into Supervised learning and Unsupervised learning.

Supervised learning

Supervised learning [34] is a method based on the idea of learning from examples represented by training data where the outputs are already known, The goal of the algorithm is to learn the relationships between the inputs and outputs and then make predictions on a separate, unseen dataset where the outputs are unknown called test data.

Formally the train data is composed of a set of n ordered pairs (x_1, y_1), $(x_2, y_2), \ldots, (x_n, y_n)$ where x_i represents a set of measurements for a single example, and y_i is the corresponding label. The test data is composed of m examples without labels $(x_{n+1}, x_{n+2}, \ldots, x_{n+m})$, and the goal of the algorithm is to predict their corresponding labels $(y_{n+1}, y_{n+2}, \ldots, y_{n+m})$.

More detailed information about Supervised learning, its applications, and algorithms can be found in [34].

Unsupervised learning

Unsupervised learning [14] is a learning paradigm where the model receives input data in the form of $(x_1, x_2, \ldots, x_n)$ without any supervised target output. In this case, the goal is to find hidden patterns or representations of the data, rather than to make predictions based on labeled examples. Some of the most common applications of unsupervised learning are clustering, dimensionality reduction, and anomaly detection.

More detailed information about unsupervised learning, its applications and algorithms can be found in [14].

Self-supervised learning

Self-supervised learning is a type of unsupervised learning where the model obtains its supervisory signals from the input data itself. The model's objective is to learn useful representations of the data by identifying patterns, structures, or relationships within it.

One common self-supervised learning technique is to mask some properties or parts of the input sequence and aims to predict the masked part or property. For example, in the NLP field, BERT [9] uses a Masked Language Model (MLM) where the model takes as input data a text sequence with some masked tokens, and the self-supervised model aims to predict the original tokens in the sequence. Another example is given in the computer vision field [19] where an autoencoder is pre-trained with masked images with the objective of reconstructing the non-visible part of the images, thus allowing the model to learn common patterns and shapes in images.

2.1.2 Perceptron

The perceptron is the basic unit of any artificial neural network, it is a linear model that makes a weighted sum of the input data $(x_1, x_2, \ldots x_n)$, a set of weights $(w_1, w_2, \ldots, w_n)$ and a bias term b, then it passes the result throws an activation function which generates the final output as shown in equation 2.1.

$$y = f(\sum_i x_i \cdot w_i + b) \tag{2.1}$$

The most common activation functions for the perceptron are the sigmoid function and the rectified linear unit (ReLU), these functions map any real

input number to a binary output as shown in equations 2.2 and 2.3 respectively, which makes this algorithm suitable for binary classification models

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

$$R(x) = max(0, z) \tag{2.3}$$

The learning process of the perceptron is based on the random initialization of the weights and bias, feeding the data into the perceptron, evaluating the output, backpropagate the error to update the weights and bias using the gradient descent algorithm, and finally repeating the process until convergence or the maximum number of iterations is reached.

Even though the perceptron algorithm has advantages such as its simplicity and computational efficiency, it also has several limitations, such as its linearity and its limitation to binary classification. In order to solve more complex problems the Multi-layer perceptron was introduced.

Multi-layer perceptron

This algorithm is born from the idea that many perceptrons can be joined to address more complex problems and computations, a Multi-layer perceptron (MLP) can be represented as a directed graph $G = (V, E)$ where the perceptrons represent the nodes in (V), and each directed edge in (E) is a communication link between the output of one neuron to the input of another one. The Multi-layer perceptron is composed of T disjoint subsets of perceptrons called layers $(V_0, V_1 \ldots, V_T)$ where $V = \cup_{t=0}^{T}(V_t)$, each layer is followed by an activation function and is classified into three types, as follows:

- Input layer (V_0): this layer is responsible for processing the input data and feeding the following layers, it consists of a set of neurons $x_1, x_2, \ldots, x_n$ where n is the dimensionality of the input data.
- Hidden layer(s) $(V_1, \ldots V_{T-1})$: These are the computational engine of the neural network, these layers are responsible for extracting features from the input data and using it to generate predictions, each neuron in the layer applies a weighted sum followed by a non-linear activation function, as shown in equation 2.1.
- Output layer (V_T): It receives the processed data from the hidden layers and generates the final output of the neural network.

The training process of an MLP is composed of two main steps forward and backward propagation, these two processes are complementary and depend on each other during training an MLP, The forward propagation calculates and saves the intermediate variables, outputs, and loss from the input layer to the output layer. Then, the backward propagation is made in the reverse order, where the algorithm calculates and saves the gradients of all the MLP parameters. Finally, these parameters are updated in order to minimize a given loss function.

The MLP is the core of deep learning algorithms, and it has been used to develop more complex algorithms, including Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), Autoencoders, and Generative Adversarial Networks (GANs). In this work, we will focus on the use of autoencoders, more specifically on Masked Autoencoders, which are algorithms that can be trained in a self-supervised way to then be fine-tuned on more specific supervised applications.

2.1.3 Autoencoders

The Autoencoder is a type of neural network architecture introduced by [22] with the objective of learning compact representations from higher-dimensional input data in an unsupervised way. The autoencoder's architecture consists of two main parts: the encoder which compresses the input data generating a latent representation, and the decoder which is useful for reconstructing the latent representation into the target data representation.

Depending on the input data and the task at hand, certain characteristics of the autoencoder need to be determined. Firstly, the type of layer to use needs to be considered. The most commonly used types of layers are:

- Convolutional layers: These are commonly used to deal with tasks involving images, such as image denoising, compression, and generation.
- LSTM layers: These are typically used to deal with sequential data, particularly time series, due to their ability to capture long-term relationships in the data.
- Transformer layers: These are particularly useful for capturing long-range dependencies in sequential data and are commonly used in NLP tasks such as language modeling and machine translation.

- Dense layers: These are primarily used for non-sequential data as they are limited in their ability to create long-range dependencies in sequential inputs.

To achieve optimal performance, a combination of the above layer types can be used. For example, dense layers are often added on top of the decoder to map the output of convolutional, transformer, and LSTM layers to the desired output data shape.

After determining the type of autoencoder to use, the main hyperparameters to decide its size are as follows:

- Latent representation size: This determines the number of neurons in the bottleneck and how much the data will be compressed.

- Number of layers: The complexity of the model is determined by the number of layers, which also affects the speed of processing. In most cases, the encoder and decoder have the same depth, but some specific applications work better with non-symmetrical architectures.

- This hyperparameter determines the number of neurons in each layer. Typically, the number of nodes in the encoder decreases layer by layer until it reaches the bottleneck, while the opposite occurs in the decoder.

Even though the autoencoder is not a new architecture, in the last years it has been quite popular given the variety and complexity of its applications, the most common ones are dimensionality reduction [37], image denoising [26], image generation [17], and anomaly detection [39].

A more detailed description about autoencoders can be found in [22].

2.1.4 Transformers

The transformer [36] is a deep learning architecture based on an attention mechanism to learn the dependencies between input and output sequences. Originally proposed for sequence data in language modeling and machine translation, it outperforms the state of the art techniques such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU).

2.1.5 Model Architecture

The transformer is composed of two main components: the encoder and the decoder. The encoder maps an input sequence $(x_1, x_2, \ldots, x_n)$ to a latent representation $z = (z_1, z_2 \ldots, z_n)$ known as encoding which captures the contextual information from the input. Then the decoder uses the latent representation z to generate an output sequence $(y_1, y_2, \ldots, y_n)$. Figure 2.1 illustrates the transformer architecture.

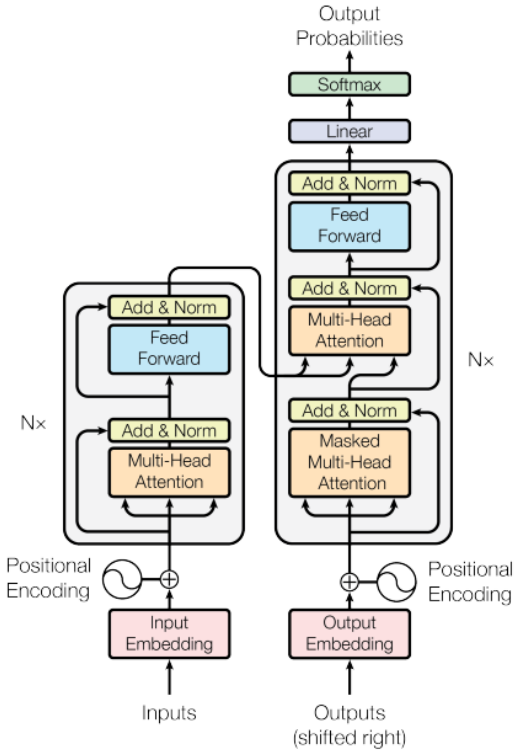

Figure 2.1. Transformer - model architecture.

Source: [36]

Encoder and Decoder

The encoder is a stack of layers, where each layer is composed of two sub-layers: a Multihead Self-Attention (MSA) mechanism that produces a set of encodings from the input, and a position-wise fully connected feed-forward network (FFN) that produces an embed of each encoding. In addition, a residual connection is used around each of the two sub-layers, followed by

layer normalization. The decoder is similar to the encoder, it consists of a stack of layers where each one is composed of a Multihead Self-Attention and fully connected feed-forward network sub-layers. The decoder also includes a third layer that performs multi-head attention on the encoder output.

Multi-head Attention

The attention mechanism was introduced for the first time in [3]. It allows the transformer to focus on the most important parts of an input sequence by weighting each part of the sequence according to its relative importance creating an attention-weighted sum. More precisely, the attention mechanism helps the transformer to effectively capture long-range dependencies at each step on the sequence. This is an improvement with respect to other architectures like RNNs where the predictions are made attending to the immediate surrounding parts of the current step. The multi-head attention is a variant of the original attention mechanism where the model attends to different parts of the sequence simultaneously by using multiple attention heads. Each head produces its own attention-weighted sum, allowing the model to learn different relationships between elements in the sequence and form a final attention output.

Positional encoding

The positional encoding allows the transformer to distinguish the order of each element in the sequence. This is important because it helps to avoid problems like permutation invariance, and improves other mechanisms of the transformer like the attention. There are several options for the positional encodings, in [36] it is used the sine and cosine functions computed as shown in 2.4.

$$\begin{aligned} PE_{(pos,2i)} &= sin(pos/10000^{2i/d_{model}}) \\ PE_{(pos,2i+1)} &= cos(pos/10000^{2i/d_{model}}) \end{aligned} \tag{2.4}$$

Where d_{model} is the dimension of the positional encodings which is the same as the embedding dimension, pos is the position of the element in the sequence and i is the dimension.

2.1.6 Visual transformers (ViT)

Presented by [11], this is an introduction to the use of transformers [36] in the computer vision field. This work was inspired by the success of the transformers in the NLP field and its objective is to apply transformers to images following the architecture of the original transformer [36] as closely as possible.

Originally the standard transformer receives as input a 1D sequence of tokens, but the images are presented in 2D and it is not straightforward how to tokenize them. The images have to be reshaped from $X \in R^{H \times W \times C}$ into a sequence of flattened 2D patches $X_p \in R^{(N \times (P^2 \cdot C))}$, as illustrated in figure 2.2.

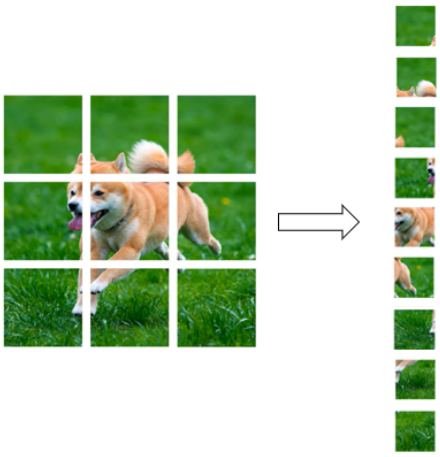

Figure 2.2. Example of patch creation and flattening process.

where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and $N = HW/P^2$ is the total number of patches, which corresponds to the effective input sequence length for the transformer.

The Transformer takes as input a latent vector of size D, which is why each image patch of resolution (P, P) is flattened and mapped to D dimensions through a trainable linear projection. The result of this step is added to the positional embedding to obtain the patch embedding, which is then processed by the transformer as described in [36].

This methodology achieves excellent results when pre-trained on large datasets such as ImageNet-21K. However, when trained on mid-size datasets like ImageNet-1K, the model's performance falls some percent point below

the state-of-the-art models of comparable size like ResNet. This drawback is explained by the lack of inductive biases of the transformers in comparison to the standard methodology in the computer vision field the Convolutional Neural Networks (CNNs).

The ViT demonstrates to reach or exceeds state-of-the-art performance on several image classification datasets, including ImageNet and CIFAR-100.

2.1.7 Masked autoencoders

A masked autoencoder (MAE) is an unsupervised learning architecture first introduced by [38], this work introduces a modification to the original autoencoder framework to integrate robustness to partially destroyed inputs. The main objective is to learn latent representations from the dependencies and regularities in the observed input. This methodology proposes an autoencoder trained on a randomly corrupted version of the input data to generate repaired version as output. As described, the input features are corrupted by randomly choosing a fixed number of components and forcing their value to 0, hiding all the information about these components to the autoencoder. The objective of the autoencoder is to recover this missing information. Figure 2.3 provides a graphical representation of the training process.

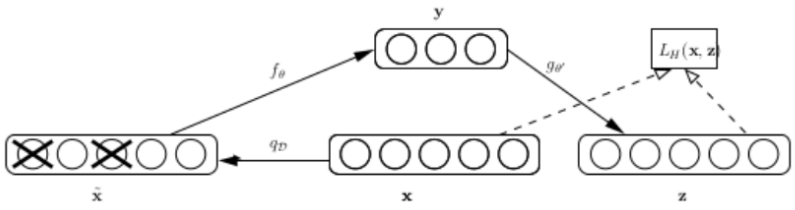

Figure 2.3. Graphical representation of the training process for a masked autoencoder.

x represents the original input, $\hat{x}$ represents the corrupted version of the input, y represents the latent representation of $\hat{x}$ generated by the encoder, the decoder attempts to reconstruct x generating z as output, $L_H(x, z)$ represents the reconstruction error. Source: [38]

The masked autoencoder has been successfully applied mainly in the field of NLP with the introduction of models like BERT [9]. In this thesis project,

we inspired our work in the applications of Masked autoencoders in the computer vision field and sound recognition, more precisely in the methodologies introduced by [19][20].

2.1.8 Masked autoencoders are scalable vision learners

ImageMAE [19] is an application of the Masked Autoencoders in the computer vision field, inspired by the success of the masked autoencoders in NLP with BERT [9], The goal of this work is to replicate the methodology used in NLP and demonstrate that masked autoencoders are scalable self-supervised learners models for computer vision.

In order to apply masked autoencoders to computer vision, it is crucial to understand the difference between text and images. Text is usually human-generated and is highly semantic and information-dense, whereas images are visual objects containing more redundant information that can be easily predicted based on context. It is important to consider this difference as it affects the amount of information that can be masked without affecting the model's performance. For example, BERT [9] has demonstrated to achieve its best results by masking only the 15% of the tokens in text, while ImageMAE [19] has been demonstrated to achieve its best results by masking a high rate (75%) of tokens, thereby reducing the redundant information in images.

In general, the proposed methodology divides images into patches, masks a high rate of them, uses the encoder to create a latent representation from only the visible patches, and finally uses the decoder to reconstruct the input image. Figure 2.4 illustrates the main idea of this work.

The main components of ImageMAE are:

- Masking strategy: The images are divided into regular, non-overlapping patches. Then a random subset of patches is masked according to a uniform distribution. As described, a high ratio of patches is masked to reduce redundancy and create tasks not easy to solve by extrapolation from visible neighbor patches.

- Encoder: Composed by a set of ViT [11], the encoder embeds patches through a linear projection with added positional embeddings. It then processes the resulting embeddings with a set of transformers to obtain the latent representation of the input. Notice that the encoder only works on the visible patches (25%), decreasing the usage of computational resources.

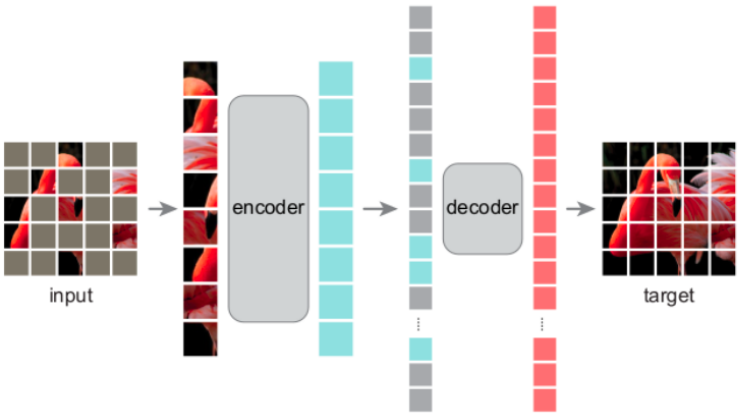

Figure 2.4. ImageMAE architecture.

The input is divided into patches and 75% of those are masked, Then the visible patches are flattened and passed through the encoder, which generates a latent representation (represented by sky-blue squares). The patches are reorganized and the mask tokens are included (represented by gray squares), the complete set of tokens is passed through the decoder, which generates patch embeddings (represented by pink squares). Finally, these embeddings are passed through a linear layer to reconstruct the original patches. source: [19]

- Decoder: The decoder is used to reconstruct the input image from the latent representation generated by the encoder. It receives as input the complete set of tokens composed of mask tokens representing missing patches to be predicted, and tokens containing the latent representation of the visible patches.

The training process of this model consists of two steps: self-supervised pre-training to learn the contextual information of the input images by reconstructing masked patches to obtain the original input. Supervised fine-tuning where the latent representations generated by the encoder are used for tasks such as object detection, semantic segmentation, and classification. In conclusion, this work demonstrates the scalability of masked autoencoders for computer vision tasks.

2.1.9 Masked Autoencoders that Listen

AudioMAE [20] is a simple extension of Masked Autoencoders ImageMAE [19] for sound recognition. It proposes a scalable framework for learning self-supervised audio representations following the encoder-decoder architecture of ImageMAE. the paper also introduces a new attention mechanism in the decoder called local attention windows, which improves the results of the autoencoder by taking advantage of the correlations in local windows of time and frequency bands in the audio spectrograms.

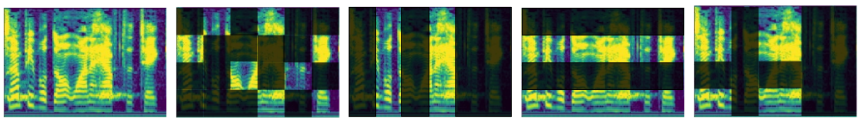

Figure 2.5. AudioMAE's masking strategies.

Source: [20]

The main components of AudioMAE are:

- Spectrogram patch embeddings: To use ViT [11], the audios are transformed into Mel-spectrograms and divided into non-overlapped regular grid patches. Similar to ImageMAE [19] the patches are flattened and embedded by a linear projection. Then the positional embedding is added to obtain the patch embed.

- Masking strategies: Based on the nature of the audio spectrograms, this work proposes two different types of masking strategies: Unstructured, random masking as made in ImageMAE [19], and Structured, where portions of time, frequency, or both are masked randomly, figure 2.5 illustrates the proposed types of masking.

  Similar to ImageMAE [19], audio spectrograms are highly redundant, then a high masking rate (80%) must be used to obtain the best results.

- Encoder: Similar to MAE [19], the encoder is composed of a stack of ViT [11], which process only the non-masked patches to create a latent representation of the audio spectrogram.

- Decoder: Composed by a stack of ViT [11]. It processes the complete set of tokens, including mask tokens that represent missing patches to be

predicted, and the latent representation of the visible patches generated by the encoder. A linear head is introduced to predict and reconstruct the input spectrogram.

Due to the nature of the audio spectrograms and their differences from the visual objects, the global self-attention in the decoder proposed in ImageMAE [19] may produce sub-optimal results. That is why they introduce a Local Attention mechanism which groups patches into local windows for self-attention, Two types of this method were proposed:

1. Shifted windows location: The attention window is shifted by 50% between consecutive transformer layers. As shown in Figure 2.6.
2. Hybrid window attention: combines global attention and local attention in order to obtain better cross-window connections.

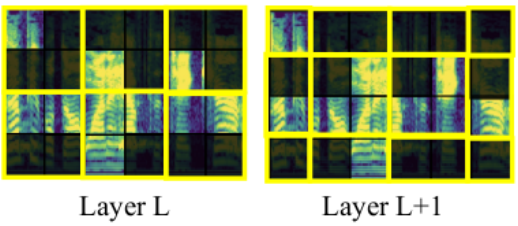

Figure 2.6. Example shifted windows location.

Source: [20]

Similar to MAE [19], the training process of AudioMAE is divided the two steps: pre-train and fine-tuning. During pre-training, the masked autoencoder learns the contextual information of the input spectrograms by reconstructing the masked patches. During fine-tuning, the encoder's knowledge is used in different tasks like audio classification, speech classification, and speech commands recognition. Unlike ImageMAE [19], AudioMAE introduces the use of mask tokens in the encoder during fine-tuning.

AudioMAE achieves three main goals. First, it achieves state-of-art performance on six audio and speech classification tasks. Second, it introduces local self-attention in the decoder, resulting in stronger representations of the input spectrograms. Third, it demonstrates that the use of masking in fine-tuning improves accuracy and reduces computational time.

In conclusion, deep learning algorithms have been widely used in different tasks, generating new state-of-the-art results. For example, in [19] and

[20], it has been demonstrated how the same autoencoder architecture has reached state-of-the-art results in tasks from different fields, such as image classification and speech recognition. These results show us the capability of masked autoencoders to achieve high performance on different types of data and applications. Therefore, in this thesis, we investigate the use of masked autoencoders in the SHM field.

2.2 Structural Health Monitoring

Structural Health Monitoring (SHM) is defined by [13] as the implementation of damage identification strategies for aerospace, civil and mechanical engineering infrastructure. This process involves the observation over time of a structure by the collection of data throws sensors or data acquisition systems that together with statistical analysis help to determine the current state of a structure.

The objective of SHM is to detect changes in the structure behavior that can be considered as damage. The damage does not mean the complete loss of the system functionality, but it evidences that the structure is not working in an optimal manner, this can grow until the point of failure where the structure operation is not acceptable to the user.

The main advantages of applying SHM methods to detect changes and damages in early times are as follows:

- Improves structural reliability and life cycle management, Helps the decision-making process about maintenance and repair, ensures the safety of the structure, and prolongs its life cycle.
- Minimize intervention and maintenance costs. It avoids large and disruptive maintenance.
- Improves understanding of the structure: By collecting and analyzing SHM data, it is possible to understand the structural behavior and how it varies in different weather or load conditions.

Chapter 3

Related work

Structural health monitoring (SHM) has been an area of growing interest during the last few years because of the increasing demand for reliable methods for evaluating the state of complex structures, such as bridges, viaducts, or buildings. This interest is motivated by the need to improve the safety of the users and also for the economic impact these technologies can have by improving the maintenance activities or avoiding the complete loss of the system functionality [13]. In the past, the SHM activities were mainly sporadic human assessment of the structures, however, currently, the application of IoT systems equipped with low-cost and self-powered sensors able to stream high-frequency data allows the continuous observation of the structures and the application of machine learning algorithms to the SHM activities, this new monitoring paradigm is known as the SHM 4.0 revolution [28].

The application of IoT systems and machine learning algorithms have been used for SHM only in the last few years, some common applications are in tasks such as image processing and vibration-based damage detection. For example, [15] proposed a methodology for autonomous post-disaster reconnaissance based on the use of low-cost unmanned aerial vehicles to capture visual data on structures affected by natural disasters, to then be analyzed with region-based convolutional neural networks to detect four different damage types. in [21] a network of distributed acceleration sensors is used to capture the floor movement, to then feed classical machine learning algorithms such as support vector machine, K-nearest neighbor, and convolutional neural networks, with the objective of identifying the status of buildings post-disaster events.

This thesis work is focused on the use of data generated by a network of SHM sensors in two applications, anomaly detection, and Traffic Load

Estimation (TLE). These applications of SHM data have been explored in previous works, as described above.

In [12] an Autoregressive Moving Average (ARMA) is used for feature extraction from high dimensional SHM vibration data coming from a cable-stayed bridge to then identify unnormal measures with a divergence analysis, this pipeline reaches a misclassification error of 1.56% demonstrating its effectiveness to detect damage.

In [2] is proposed a one-dimensional convolutional neural network that runs directly on the raw vibration data, this approach demonstrates to be efficient to identify structural damages on a benchmark dataset.

The work on [5] proposes a framework to monitor vehicles passing on a viaduct, this methodology works with three axial data captured by SHM accelerometers. Anomaly detection algorithms were used to detect vehicles passing the viaduct, while a linear regression was used to estimate the speed of the passing vehicles.

Finally, in [40] is proposed a methodology to obtain traffic information from pavement vibrations captured by a vibration-based in-field pavement monitoring system, the data was used to feed a three-layer artificial neural network and a k-means++ cluster analysis with the objective of monitoring the vehicles speed, axle spacing, driving direction, location of the vehicles, and traffic volume.

The following part of this section will focus on the analysis of two applications developed on vibration-based SHM data, we put particular interest in these studies because they use a data source similar to the one we use in our experiments allowing us to use the results of these applications as baselines for our research.

3.1 Anomaly detection in Structural Health Monitoring

This section is based on the work presented in [29], this work introduces a pipeline for anomaly detection based on vibration data collected by SHM sensor nodes, its main objective is to develop an automatic anomaly detection pipeline that does not require communication, processing and storing raw sensor data in the cloud.

We put particular interest in this paper because it addresses the anomaly detection problem on the same dataset as our application. This is important because it gives us a direct baseline to evaluate the performance of our

proposed methodology.

Currently, the majority of works about anomaly detection on SHM are based on simulated data as for example [24][30], this work [29] based its analysis on real vibration data collected from a viaduct located in Italy. The analyzed viaduct underwent technical intervention to strengthen its structure, which generated two different sets of data from the SHM sensors: vibration data before and after the intervention. The data after the intervention represents the viaduct functioning optimally, while the data before the intervention represents a change in the viaduct structure caused by events such as earthquakes or lack of maintenance. This data was labeled as "anomalies" due to the degradation of the viaduct in comparison to the optimal scenario.

The data collection process in [29] was based on a vibration-based SHM system that consisted of five sensors to detect damages and monitor the health condition of the viaduct.

The methodology proposed in [29] is composed of three steps: Anomaly detection pipeline, algorithm phases, and deployment. In this thesis, we will focus our analysis on the first two steps, which are relevant to our scope.

3.1.1 Anomaly detection pipeline

The proposed anomaly detection pipeline consists of three sequential steps: data pre-processing, signal reconstruction, and anomaly detection. In the data pre-processing phase, the vibration data in the z-axis is divided into non-overlapping windows. An energy-based filter is then applied to discard all windows containing only sensor white noise. The energy of each window is calculated using the equation 3.1. A threshold is defined based on the observed windows energy of the training data, and windows with energy lower than the threshold are discarded to only provide informative data for the anomaly detection algorithms. This work demonstrates that using this energy filtering improves the accuracy of the algorithms.

$$E = \sum_{i=1}^{W_d} X_i^2 \tag{3.1}$$

In the signal reconstruction phase, three well-known compression-decompression algorithms are compared: Principal Component Analysis (PCA), a fully connected autoencoder, and a convolutional autoencoder. The objective of these algorithms is to compress the raw vibration data into a latent representation, and then decompress the representation to reconstruct the input signal.

After training each algorithm solely with normal data, the anomaly detection criterion is based on the assumption that the algorithms should be able to reconstruct normal data with a low reconstruction error, while anomalies should generate larger reconstruction errors. To detect anomalies, a statistical threshold is established to achieve only 0.01% of statistical false positive errors.

3.1.2 Algorithm Phases: Train, Detect, Re-Train

This work proposes an innovative methodology for implementing an anomaly detection algorithm and updating it over time. The first step is the training phase, where the algorithm is selected, parameters and hyper-parameters are tuned, and the final model is trained. Next, in the detection phase, the trained model is used for online detection. Finally, the re-training phase updates the model over time to adapt to changes in the signal dynamics

3.1.3 Results

The proposed methodology in [29] has shown that the PCA-based anomaly detection algorithm is the best performing, achieving an accuracy of 98.8%. This algorithm was compared to other state-of-the-art algorithms by applying those methodologies to the data in this application and was found that the proposed methodology based on the PCA algorithm outperforms all of them.

In conclusion, [29] proposes a methodology for anomaly detection based on PCA able to achieve an accuracy of 98.8% to identify anomalies, it proposed an energy-based threshold of the raw accelerometer data which proves to increase the accuracy and decrease the consumption of computational resources. The methodology is also flexible and can be updated over time to accommodate changes in the signal's dynamics

3.2 Traffic Load Estimation from Structural Health Monitoring sensors

This section describes the work presented on [6], which introduces a methodology for the TLE problem using supervised learning based on SHM-sensor data. Similar to the data collection process described in [29], this study utilizes as its data source an SHM sensor network installed on a viaduct in Italy,

the novelty in this application is the introduction of a set of labels describing the count of vehicles traversing a specific lane on the viaduct.

The labels were generated by recording a video of a section of the viaduct and then using this information to generate the labels for the acceleration data captured by the SHM sensors. The main objective of this study is to estimate the count of vehicles passing through the viaduct in a given time interval. However, in this study was decided to employ separate estimators to count light vehicles and heavy vehicles.

In [6], the SHM network was composed of 7 sensors measuring the vibrations in the acceleration axis x, y, z. The methodology begins with dividing the vibration data into windows of length T_{win} and assigning the corresponding labels. A feature extraction procedure is applied to each one of the windows, which involves calculating a set of 12 basic statistical features reported in Table 3.1 for each acceleration axes of each sensor. Finally, a correlation-based filtering technique was applied to select only the most relevant features and decrease the complexity of the regression models.

Name	Description
mean	Mean acceleration
std	Acceleration standard deviation
min	Minimum acceleration sample
max	Maximum acceleration sample
med	Acceleration median
kurt	Kurtosis coefficient
skew	Skewness index
rms	Root Mean Square value
sabs	Sum of the absolute values
eom	Count of elements larger than the mean
ener	Acceleration energy
mad	Median Absolute Deviation

Table 3.1. Acceleration features extracted for each axis.

Source: [6].

Six different regression algorithms were trained to predict the count of vehicles on each window, As mentioned, this process was divided for the light and heavy vehicles resulting in the development of two separate regressors. The trained algorithms are reported in Table 3.2 together with the best-performing hyperparameters.

Model	Hyperparameters
Linear Regression	-
Random Forest	Max Depth = 200, Trees = 30
K-Nearest Neighbors	k = 7
Support Vector Regressor	Layers = 3, Neurons = 100
Multi-Layer Perceptron	Kernel = RBF, C = 10.0
LeNet5	Default of [23]

Table 3.2. Regression algorithms used on [6].

In this application, the Support Vector Regressor achieves the best performance in both categories reaching an MAE of 0.47 for light vehicles and 0.21 for heavy vehicles.

According to [6] this is the first time that the TLE problem is addressed in a supervised way with SHM vibration data. The study demonstrates how adding labels to the vibration data results in a significant improvement in the performance of the models in comparison with unsupervised applications.

We are particularly interested in this paper because it applies to a dataset that comes from a similar data source as our application. This is important because it enables us to replicate these models on our data and have a direct point of comparison to evaluate the performance of our proposed methodology.

Chapter 4

Masked Autoencoders on Structural Health Monitoring

The objective of this thesis is to apply the methodologies proposed by MAE [19] and AudioMAE [20] to SHM vibration data for two applications: anomaly detection and TLE. For anomaly detection, the aim is to detect damages in the viaduct structure. For TLE, the goal is to estimate the load of the structure at a given time. Our methodology follows the two-stage approach outlined in [19][20]. The first stage is a pre-training phase, where the autoencoder learns latent representations from the dependencies and regularities of the SHM data by training the autoencoder to reconstruct the input data in a self-supervised way. The second stage is a fine-tuning phase, where the pre-trained autoencoder is modified and further trained for a specific application, such as TLE.

This chapter focuses on describing in detail the pre-training phase of our masked autoencoder. The chapter is structured as follows: In section 4.1, we describe the data source and how the data is preprocessed to feed the autoencoder. In section 4.2, we describe the base autoencoder architecture including the main components such as the encoder, decoder, and masking strategy. Finally, in section 4.3, we provide details on the pre-training process, including weight initialization, optimizer selection, and learning rate determination.

Figure 4.1 depicts the complete architecture of our masked autoencoder described in the following sections.

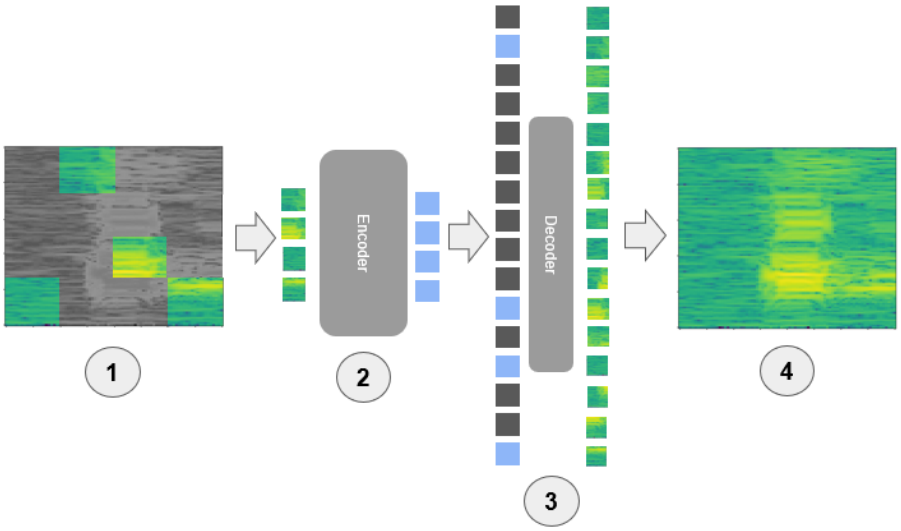

Figure 4.1. Complete Masked Autoencoder architecture.

The illustration depicts an architecture consisting of four parts. 1) the Input masked spectrogram is shown, in this a large subset of patches is masked to reduce redundancy. 2) the encoder receives only the remaining visible patches and generates a latent representation for them. 3) The mask tokens are introduced before the decoder, and the full set of patches is ordered with the positional encoding to give the decoder information about the position of each patch. 4) the decoder reconstructs the original spectrogram by predicting all the pixels. This architecture is used during the pre-training phase, and during fine-tuning, the decoder is discarded while the encoder is used for specific tasks.

4.1 Data

4.1.1 Data source

The data analyzed in this thesis was collected by an SHM network composed of 3-axial Micro Electro-Mechanical System (MEMS) analog accelerometers installed on a viaduct in Italy. The viaduct is divided into 18 spans of length ranging from 32 to 35 meters. Each span is monitored by 6 sensors installed on prestressed tendons used to strengthen the structure giving a total of 108 sensors monitoring continuously the viaduct. Figure 4.2 represents the installation of the sensors on spans 2 and 3 of the viaduct, and the distance between them.

The main objective of the SHM sensor network is to monitor the viaduct

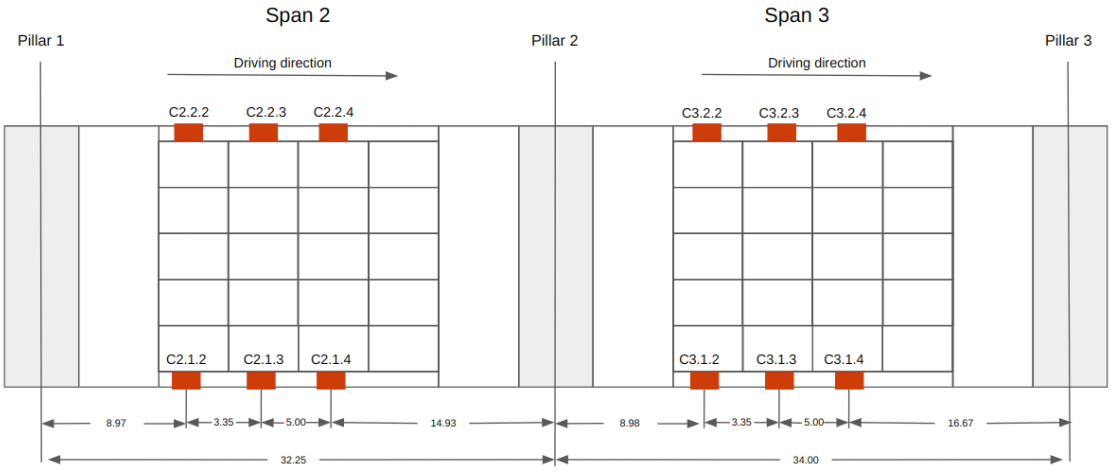

Figure 4.2. Graphical representation of the viaduct's spans 2 and 3.

The position of the sensors is represented with red squares with their corresponding identification, all the distances are reported in meters.

structure by measuring continuously the vibration dynamics. Each of the sensors in the network captures the linear acceleration in three directions (x, y, z) at a sampling frequency of 100Hz. It is important to note that our methodology involves utilizing data from pre-existing sensors, which enables us to repurpose them for our purposes without incurring additional installation or modification costs. However, it should be noted that the position and installation of these sensors may not be optimal for detecting anomalies or performing TLE tasks.

4.1.2 Data preprocessing

For the aim of this analysis, we only use the acceleration in the z-direction as it is perpendicular to the ground surface. The signal is divided into overlapping windows of duration (T_{win}) and stride (T_{stride}). Following the methodology of [29], an energy filter is applied to discard non-informative windows containing only the white noise of the viaduct and preserve windows with disturbances generated by vehicles crossing the viaduct. The energy of each window is extracted following equation 3.1 and compared with an energy threshold. Any window with an energy lower than the threshold is discarded.

After applying the Masked-autoencoder proposed in the paper by [20], we

transform the one-dimensional vibration signal into two-dimensional spectrogram using the following procedure: First, we extract the constant component of each window by subtracting the mean value of the signal from each data point. Then, we compute the Power Spectral Density (PSD) spectrogram by dividing the window into overlapping segments and applying equation 4.1 to each segment, where $PSD(f)$ is the power spectral density at frequency f, $X(f)$ is the Fourier transform of the signal, N is the total number of samples, and Fs is the sampling frequency. The resulting power spectra are stacked vertically to form a 2D array, which is then subjected to min-max normalization. The preprocessing pipeline is summarized in Figure 4.3.

$$PSD(f) = \frac{|X(f)^2|}{N * Fs^2} \tag{4.1}$$

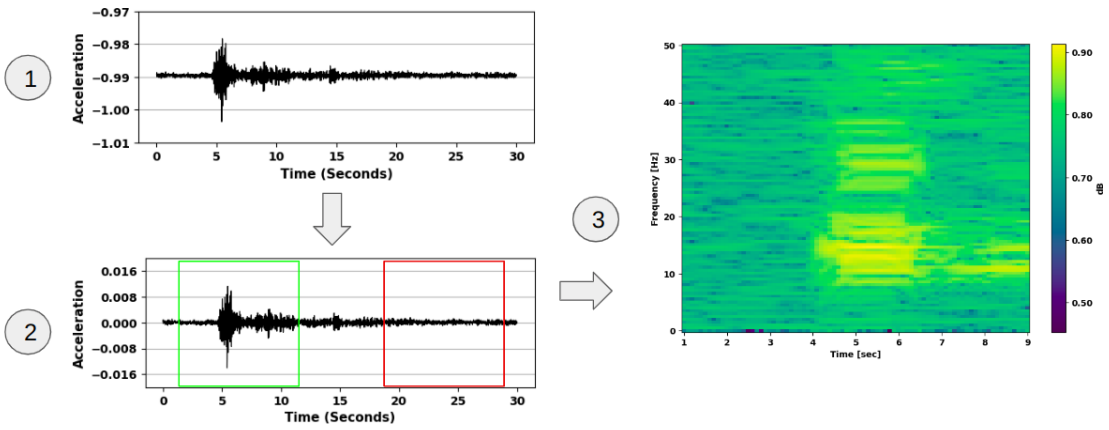

Figure 4.3. Signal preprocessing pipeline.

Complete preprocessing pipeline: 1) Original signal. 2) The constant component of the signal is deleted, and the remaining signal is filtered, the green square represents an informative window, which contains useful signal information, in contrast, the red square represents a window composed of white noise, which contains no useful signal information. 3) The normalized PDS spectrogram is generated from the informative window signal.

4.2 Masked Autoencoder

4.2.1 Patch embeddings

Following the methodology in [20], we divide each spectrogram into non-overlapped regular grid patches of patch size P. We then flatten each patch and embed it using a linear projection layer. To incorporate positional information into the patches, we employ sinusoidal positional embeddings, as shown in Equation 2.4.

4.2.2 Masking

Based on the methodology proposed in [20] and its results, we adopt the unstructured masking strategy. This method randomly selects the patches to be masked according to a uniform distribution and eliminates all information from those patches. An example of this masking strategy is shown in Figure 2.5 b. To eliminate redundancy from the spectrograms, we adopt a high masking ratio of 80%, as recommended in [20]. Listing 4.1 shows the python implementation of the masking strategy.

```
def random_masking(self, x, mask_ratio):
    """
    Perform per-sample random masking by per-sample shuffling
    Per-sample shuffling is done by argsort random noise.
    x: [N, L, D], sequence
    """
    N, L, D = x.shape  # batch, length, dim
    len_keep = int(L * (1 - mask_ratio))

    noise = torch.rand(N, L, device=x.device)  # noise in [0,
    1]

    # sort noise for each sample
    ids_shuffle = torch.argsort(noise, dim=1)  # ascend:
   small is keep, large is remove
    ids_restore = torch.argsort(ids_shuffle, dim=1)

    # keep the first subset
    ids_keep = ids_shuffle[:, :len_keep]
    x_masked = torch.gather(x, dim=1, index=ids_keep.
   unsqueeze(-1).repeat(1, 1, D))

    # generate the binary mask: 0 is keep, 1 is remove
    mask = torch.ones([N, L], device=x.device)
```

```
22      mask[:, :len_keep] = 0
23      # unshuffle to get the binary mask
24      mask = torch.gather(mask, dim=1, index=ids_restore)
25
26      return x_masked, mask, ids_restore
```

Listing 4.1. Python implementation of the masking strategy

4.2.3 Encoder

According to [20], we construct our encoder as an n-layer stack of ViT [19] applied to the visible patches, The value of n is a hyperparameter that we will set in the following sections, and it will depend on each application.

```
1  def forward_encoder(self, x, mask_ratio):
2      # embed patches
3      x = self.patch_embed(x)
4
5      # add pos embed w/o cls token
6      x = x + self.pos_embed[:, 1:, :]
7
8      # masking: length -> length * mask_ratio
9      x, mask, ids_restore = self.random_masking(x, mask_ratio)
10
11     # append cls token
12     cls_token = self.cls_token + self.pos_embed[:, :1, :]
13     cls_tokens = cls_token.expand(x.shape[0], -1, -1)
14     x = torch.cat((cls_tokens, x), dim=1)
15
16     # apply Transformer blocks
17     x = self.encoder(x)
18     x = self.norm(x)
19
20     return x, mask, ids_restore
```

Listing 4.2. Python implementation of the forward process for the encoder

Listing 4.2 shows the Python implementation of the forward process for the encoder. The function receives as input the normalized patches. In lines 3 and 6, the embedding is generated by adding the patch and positional embeddings. Then, the masking process described in Listing 4.1 (which masks a percentage of the input tokens) is called. The classification (cls) token and its positional embedding are added, and in line 14, the cls token is concatenated to the tensor of tokens. Finally, in line 17, the tokens are passed through the encoder, and in line 18, the encoder output is normalized.

4.2.4 Decoder

According to [20], we design our decoder as an n-layer stack of ViT [11]. The decoder takes as input the set of encoded patches and a set of masked tokens that represent the missing patches to be predicted. To provide the decoder with positional information about the visible and masked tokens, we assign positional encodings using the equation 2.4. Finally, we add a linear layer at the top of the decoder to reconstruct the input spectrograms.

```
1  def forward_decoder(self, x, ids_restore):
2
3      # embed tokens
4      x = self.decoder_embed(x[:, 1:, :])
5
6      # append mask tokens to sequence
7
8      mask_tokens = self.mask_token.repeat(x.shape[0],
       ids_restore.shape[1] - x.shape[1], 1)
9      x_ = torch.cat([x, mask_tokens], dim=1)  # no cls token
10     x = torch.gather(x_, dim=1, index=ids_restore.unsqueeze
       (-1).repeat(1, 1, x.shape[2]))  # unshuffle
11
12     b, l, c = x.shape
13
14     assert l == self.grid_h * self.grid_w, "input feature has
       wrong size"
15
16     # add pos embed
17     x = x + self.decoder_pos_embed
18     x = x.view(b, self.grid_h, self.grid_w, c)
19     # apply Transformer blocks
20     for blk in self.decoder_blocks:
21         x = blk(x)
22
23     x = rearrange(x, 'b h w c -> b (h w) c')
24     x = self.decoder_norm(x)
25
26     # predictor projection
27     x = self.decoder_pred(x)
28
29     return x
```

Listing 4.3. Python implementation of the forward process for the decoder

Listing 4.3 presents the Python implementation of the decoder's forward process. The function takes the encoder output as input. In line 4, it generates the decoder embeddings by applying a linear projection layer that maps

the encoder output to the expected shape of the decoder input. Then, from lines 8 to 10, the function generates the mask tokens, concatenates them to the tokens, and reorders them according to their original position. In lines 17 and 18, the function adds the positional embeddings, and the resulting tensor is reshaped to fit the decoder's expected input shape. In lines 20 and 21, the function passes the tokens through the decoder blocks, and the output is reshaped and normalized in lines 23 and 24. Finally, in line 27, a linear projection layer maps the decoder output to the shape of the input image, generating the reconstructed spectrogram.

4.2.5 Local attention

According to the findings in [20], spectrograms exhibit different characteristics and relationships than images. Because spectrograms are more sensitive to the position and scale of their components, these factors can directly affect the represented sound. Following this reasoning, our work complements the global attention of the transformers with the local attention mechanism proposed in [20]. As explained in section 2.1.9, this mechanism groups patches into self-attention windows during decoding, allowing for more precise modeling of the relationships between components within the spectrogram.

We adopt the Shifted Window Location Self-Attention (SWLSA) mechanism introduced by [20]. This mechanism groups consecutive patches in the spectrogram to generate local attention and shifts the groups 50% in the top-left direction between consecutive transformer layers. Figure 2.6 provides a visual representation of this mechanism.

4.2.6 Reconstruction target

During the pre-training phase, the masked autoencoder is trained to reconstruct the input spectrogram by predicting each pixel from the masked patches. We use the Mean Square Error (MSE) as show in equation 4.2 as a loss function to measure the reconstruction error between the input and output spectrograms. Following [20] we calculate the reconstruction loss only on the masked patches.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{4.2}$$

4.3 Pre-training configurations

In this section, we will describe the main configuration used for the masked autoencoder during the pre-training phase. These configurations are shared by both applications and are based on the optimal values found on the masked autoencoder applications discussed in [19] and [20].

4.3.1 Weight initialization

Providing a good starting point for the optimization process is critical to the performance of the autoencoder. When weights are randomly initialized, the values may be too small or too large, leading to vanishing or exploding gradient problems, respectively.

In this work, we apply the Xavier uniform initialization method [16]. This method scales the initial weights based on the number of input and output neurons in the layer. The idea is to keep the variance of the activations and gradients approximately the same throughout the network during training.

The weights are sampled from a random uniform distribution that is bounded within the range r generated by equation 4.3, where n_i is the number of input neurons to the layer and n_{i+1} is the number of output neurons from the layer.

$$r = \pm \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}} \tag{4.3}$$

4.3.2 Optimizer

The optimizer helps to update the weights and biases in each epoch according to the direction of the steepest descent of the loss function. In this application, we apply the AdamW optimizer [25] that calculates an adaptive learning rate depending on the magnitude of the gradient. In summary, the learning rate is reduced for weights with large gradients and increased for weights with small gradients, allowing the optimizer to take larger steps in flatter regions and smaller steps in steep regions.

The optimizer contains an optimizer momentum, which is an extension useful to accelerate the convergence towards the minimum loss function. This is achieved by adding a fraction of the previous update vector to the current update vector to increase the step size in the relevant direction and avoid oscillation.

$$
\begin{aligned}
v_t &= \beta_1 v_{t-1} + (1-\beta_1) g_t \\
s_t &= \beta_2 s_{t-1} + (1-\beta_2) g_t^2 \\
\hat{v}_t &= \frac{v_t}{(1-\beta_1^t)} \\
\hat{s}_t &= \frac{s_t}{(1-\beta_2^t)} \\
\theta_{t+1} &= \theta_t - \alpha \frac{\hat{v}_t}{\sqrt{\hat{s}_t} + \epsilon}
\end{aligned}
\tag{4.4}
$$

The momentum is calculated following the equations ref 4.4, where β_1 and β_2 are hyperparameters that control the decay rates of the moving averages, g_t is the gradient at iteration t, v_t and s_t are the moving averages of the gradients and squared gradients, respectively. $\hat{v}_t$ and $\hat{s}_t$ are their bias-corrected versions. θ_t is the parameter vector at iteration t. α is the learning rate, and ϵ is a small value for numerical stability. In our implementation, we set the values for β_1 and β_2 to 0.9 and 0.95, respectively.

4.3.3 Learning rate

The learning rate is the hyperparameter that controls how much the weights of the model are updated with respect to the loss gradient during training. In our implementation, we set the base learning rate to 0.0002. However, we use a warm-up procedure to gradually increase the learning rate from 0 to the base value. This procedure helps the model to adjust the initial weights before making large updates. In our implementation, we set this value at 40 epochs.

Chapter 5

Anomaly Detection for Bridge Degradation Tracking

Anomaly detection [7] is the process of identifying patterns, data points, and events that do not conform to the expected behavior in a given context. These techniques have been successfully applied in various fields, including cybersecurity [1], fraud detection [35], and monitoring of industrial systems [27]. The most common anomaly detection techniques are based on statistical methods [33], machine learning algorithms [29], time series analysis [32], and deep learning architectures such as autoencoders [39].

Although anomaly detection appears to be a standard classification problem, applying common classification algorithms such as random forest or neural networks is often ineffective in this task for several reasons. First, it is usually impossible to obtain real anomaly data for sampling. For example, in our case, it is not feasible to sample anomaly data from a structurally sane bridge. Second, classification algorithms may not be suitable for detecting anomalies because they do not follow regular patterns or have characteristics that group them into predefined categories. Thus, applying classification algorithms may not be optimal given the difficulty of creating balanced datasets that clearly represent normal and anomaly data. For this reason, anomaly detection problems rely on unsupervised techniques such as clustering (grouping data points together based on similarity) [31], density-based methods (identifying regions of low density in data) [8], and distance-based methods (measuring the similarity or dissimilarity between data points) [18].

As mentioned in section 2.1.3, autoencoders are one of the most used techniques for anomaly detection. This is because autoencoders can be trained in unsupervised or self-supervised ways, which is particularly useful in the context of anomaly detection where labels for anomaly data are not available. The basic idea behind using autoencoders for anomaly detection is that they can learn to reconstruct patterns from normal data, but may fail to reconstruct anomalies, resulting in a higher reconstruction error. Therefore, it is straightforward to detect anomalies by setting a threshold on the reconstruction error.

Following the above reasoning, In the next sections, we describe the application of the masked autoencoder introduced in section 2.1.7 for SHM anomaly detection. The rest of this chapter is structured as follows: we describe the composition and division of the dataset, complement the data preprocessing pipeline introduced in section 4.1.2, introduce the architecture of the autoencoder and the different architectures tested, and finally, introduce the anomaly detection pipeline. Figure 5.1 illustrates the proposed pipeline explained in the following sections.

5.1 Data

5.1.1 Dataset composition

As stated in Section 4.1, the dataset analyzed in this study was collected by a Structural Health Monitoring (SHM) sensor network comprising 3-axial MEMS accelerometers installed on a viaduct located in Italy. The network included 70 sensors distributed along the viaduct, which measured linear acceleration along the x, y, and z axes at a sampling frequency of 100 Hz. Figure 5.2 provides an example of the installation of MEMS sensors in a viaduct. Similar to [29], the viaduct was subjected to a planned maintenance procedure, which allowed for the collection of acceleration data before and after the intervention. The data collected after the intervention represents the viaduct functioning optimally, while the data collected before the intervention are considered anomalies due to the degradation of the viaduct in comparison to the optimal scenario. The available dataset consists of fifteen days of data, divided as follows:

- Training set: Four days after the intervention.
- Validation set: Three days after the intervention.

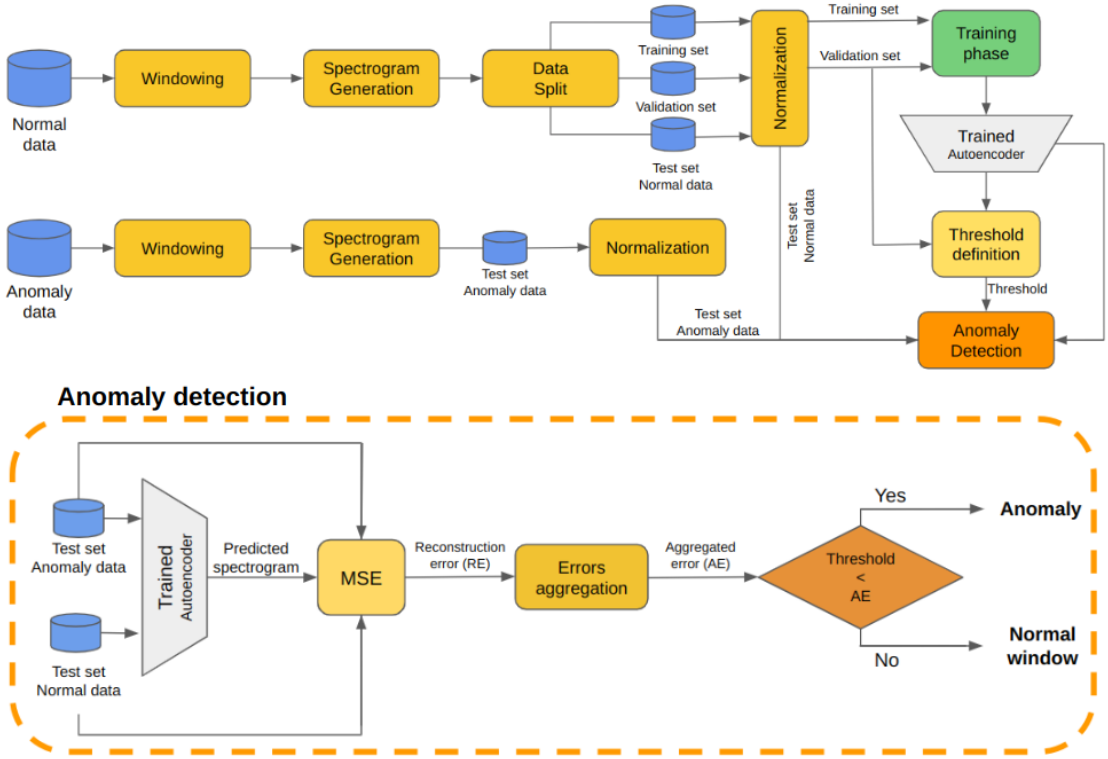

Figure 5.1. Proposed anomaly detection pipeline.

At the top of the image, the complete pipeline is described. The raw datasets for normal and anomaly data are windowed and used to generate spectrograms. The normal data is divided into training, validation, and test sets, which are then normalized. The autoencoder training process uses the training and validation sets, resulting in a pre-trained autoencoder. The pre-trained autoencoder is then used to calculate reconstruction errors on the validation set, which generates a threshold. Finally, the pre-trained autoencoder, threshold, normalized test set, and anomaly data are used in the anomaly detection process.
At the bottom of the image, the anomaly detection process is described in detail. The normal and anomaly spectrograms are passed through the autoencoder in a reconstruction task, generating the reconstruction error MSE. The reconstruction errors are aggregated as described in Section 5.3. The aggregated error is compared to the calculated threshold; if the error is greater than the threshold, the window is considered an anomaly; otherwise, it is considered normal.

- Test set for normal data: Four days after the intervention.

- Test set for anomaly data: Four days before the intervention.

Figure 5.2. Example of MEMS sensors installed on a viaduct.

Source: [6]

It is possible to notice how even if we have the availability of a dataset representing anomalies, we follow the pipeline explained above based on self-supervised learning. We choose this methodology because we aim to demonstrate that our masked autoencoder can be successfully applied in a general anomaly detention scenario where there is no availability of data representing anomalies.

5.1.2 Data preprocessing

The data preprocessing pipeline is the first step of our framework, it follows the process described in section 4.1.2. In this application, we aim to reconstruct the spectrogram of the signal generated in the z-axis for a vehicle crossing the viaduct, for this reason, the dataset was divided into overlapped windows of length $T_{win} = 10s$ and stride $T_{stride} = 1s$.

The resulting windows of dimension (1000,1) are transformers into PSD spectrograms using a $frame_{Lenght} = 198$ samples and a $step_{Lenght} = 10$ samples. The resulting spectrograms are 2D arrays of shape (100, 80). Figure 5.3 A shows some examples of the obtained spectrograms.

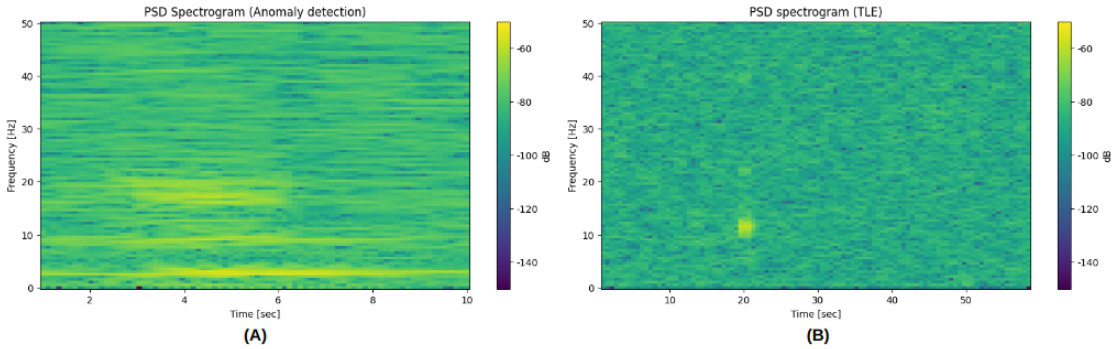

Figure 5.3. Example of generated spectrograms for Anomaly detection and TLE.

The illustration depicts an example of the spectrograms used for the different tasks. A) An example of the spectrograms used during the anomaly detection application is shown, which represents a vehicle crossing the viaduct. B) example of the spectrograms used during TLE is shown, which represents a vehicle crossing the viaduct at time step 20.

5.2 Masked-autoencoder settings

5.2.1 Patch embeddings

The 2D spectrograms are divided into non-overlapped regular grid patches of patch size $P = 5$, generating a total of 320 patches.

5.2.2 Masking

Using a masking rate $Mrate = 0.8$, in total 256 patches are randomly masked, and only 64 are visible for the autoencoder.

5.2.3 Autoencoder architecture

In the previous applications of masked autoencoders on [19][20] the optimal architecture is composed as follows:

- Encoder depth: 12 ViT layers
- Decoder depth: 16 ViT layers
- Encoder embedded dimension: 768

- Decoder embedded dimension: 512

However, the applications on [19][20] are on more extensive and diverse datasets like ImageNet-21K and Google Audioset. Because of this, we experiment with different architectures in order to find the smallest architecture able to reconstruct accurately the spectrograms. The compared architectures are summarized in Table 5.1.

Architecture	Encoder depth	Decoder depth
Small	3 Layers	4 Layers
Medium	6 Layers	8 Layers
Large	12 Layers	16 Layers

Table 5.1. Tested architectures for the pre-training phase.

Following the recommendations from [19][20], none of the proposed architectures in Table 5.1 have symmetrical encoder and decoder depths. While most autoencoder applications use symmetrical architectures, masked autoencoders are a special case because the encoder processes less information than the decoder. In this application, the encoder must create a latent representation from only 64 visible patches, whereas the decoder must reconstruct a complete image with 320 patches.

5.2.4 Autoencoder objective

As mentioned in section 4.2.6, the objective of our autoencoder is to minimize the mean squared error (MSE) 4.2 between the predicted and input spectrogram, averaged over the masked patches. The reconstruction error in the test set will then be used to identify anomalies, as explained in the next section.

5.3 Anomaly detection

This section describes the proposed anomaly detection pipeline illustrated in Figure 5.1. It starts taking the reconstruction errors as input and aims to predict whether the viaduct is operating optimally or if there is a change in its expected behavior. This is a critical process because an incorrect prediction could result in a complete loss of functionality or costly maintenance processes.

The anomaly detection pipeline is composed of two stages as described below:

- To ensure that our pipeline can handle outliers, we aggregate the reconstruction errors into windows and calculate the average error for each window. By aggregating the errors, we can define a threshold and determine whether a window is normal or anomalous. We propose two different strategies for aggregation:
  - Aggregation by amount: This strategy aggregates the reconstruction error of n consecutive reconstruction errors. We explore aggregation windows ranging from 1 (no aggregation) to 500 consecutive reconstruction errors.
  - Aggregation by time: Following the methodology proposed in [29], this strategy aggregates reconstruction errors contained in consecutive time windows of duration T_{sec} seconds. To ensure comparability with [29], we explore windows ranging in duration from 1 minute to 4 hours.
- Threshold definition: The threshold is statistically derived. First, all the reconstruction errors in the validation set are calculated, then they are aggregated as mentioned in the above section. And finally, following the methodology on [29] the threshold is defined as the mean of the aggregated MSEs plus n times the standard deviation ($th = \mu + n \times \sigma$). After determining the threshold, it is applied to partition the test set into anomalous and normal data. We explore three different values for n, and the results are exposed in the following sections.

Chapter 6

Traffic load estimation with Masked Autoencoders

Traffic load estimation (TLE) is the process of predicting the amount of traffic traveling over a network at a particular time. In the context of structural health monitoring (SHM), TLE is used to determine the quantity, frequency, and weight of vehicles passing over a structure. This process is essential in understanding traffic behavior, identifying trends and patterns, and supporting planning processes for maintenance activities, interventions, or traffic regulation.

Depending on the objective and availability of data, this problem can be framed in different ways. For example, in [6], the problem was addressed as a regression problem, and classical machine learning algorithms were used to estimate the number of vehicles, as mentioned in section 3.2. Another approach was proposed in [10], where the objective was to detect the presence or absence of a vehicle at a freeway exit at a given time. In this case, the problem was framed as a binary classification problem and addressed using a Gradient Tree Boosting algorithm.

We propose to approach the application as a regression problem. Our objective is to estimate the number of vehicles crossed on a viaduct at a given time. The following sections describe how our masked autoencoder introduced in chapter 4 can be used to address a regression problem. Figure 6.1 illustrates the proposed pipeline explained in the following sections.

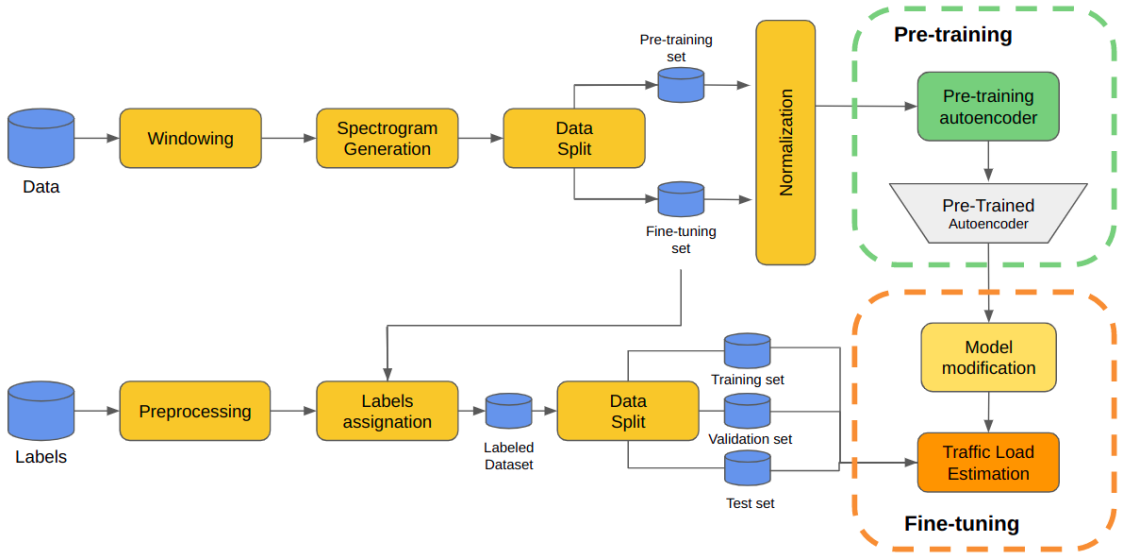

Figure 6.1. Proposed traffic load estimation pipeline.

The TLE pipeline starts by dividing the raw dataset into overlapping windows, which are used to generate spectrograms. The spectrograms dataset is then split into pre-training and fine-tuning sets and normalized. The pre-training set is utilized to pre-train the autoencoder, resulting in a pre-trained autoencoder. After completing the pre-training phase, the fine-tuning process commences by preprocessing the dataset from the scale with labels. The labels are assigned to the fine-tuning set, resulting in a labeled dataset, which is divided into training, validation, and test sets. Finally, the pre-trained autoencoder is modified by discarding the decoder and attaching a neural network on top of the encoder. This new architecture is further trained with the labeled dataset in the fine-tuning phase.

6.1 Pre-training and Fine-tuning

In order to address this regression problem with our masked autoencoder, the training process must be divided into two stages, as follows:

- Pre-training: The masked autoencoder is trained in a self-supervised way to learn latent representations from the SHM vibration data, as described in chapter 4.
- Fine-tuning: Once the pre-training phase is finished, the architecture of the autoencoder is modified to make it compatible with the new task. First, the decoder is discarded since in the new task the reconstruction

of the spectrograms is not necessary. Second, an MLP is added on top of the encoder to predict the number of vehicles. Finally, the new architecture is trained again in a supervised way.

This approach is widely used given it allows to train models on a large general dataset to learn general features of the data, to then further train the model again in a more specific task, with a smaller dataset to learn more specific features. This approach has two main advantages. First, it allows to adapt deep architectures like imageMAE, to achieve state of art results on tasks with a small dataset with labels like ImageNet1K [19]. Finally, once a model is pre-trained on a general dataset, it is possible to fine-tune it quickly on different tasks given it is not necessary to start from scratch, saving training time and computational resources. This training process has been applied in previous applications of masked autoencoders, and it has been proven to enhance the performance of models in tasks like image classification [19], audio recognition [20], and text classification [9].

6.2 Dataset composition

In this application, the dataset is composed of two data sources as described below:

- SHM sensors data: The SHM sensor data was collected using a network of sensors installed on a real viaduct located in Italy, similar to the dataset described in section 5.1.1.
- Log of vehicles: It was collected by a scale located 600m before the viaduct and registered the time, weight, and speed of all the vehicles passing the viaduct.

Refer to Table 6.1 for an example of the data generated by the scale. The dataset is composed of the following features: ID, which is a consecutive identification of the vehicles crossing the scale; Time, which saves the time in which the vehicles cross the scale with a precision of 1 minute; Weight, which saves the weight of each vehicle in kilograms; and Speed, which saves the speed of the vehicles crossing the scale expressed in Kilometers per hour (Km/H).

Both datasets were collected synchronously in a continued registration of 48h. However, during the pre-training phase, only non-labeled data will be

passed to the autoencoder, while during fine-tuning the model will be trained with labeled data. The division of the dataset will be as follows:

- Pre-training set: 24h of SHM sensor data without labels.
- Fine-tuning train set: 12h of SHM sensor data with labels.
- Fine-tuning evaluation set: 6h of SHM sensor data with labels.
- Fine-tuning test set: 6h of SHM sensor data with labels.

Notice that even with if we have the availability of labels for all the SHM vibration data, our objective is to demonstrate that in this context a pre-training phase on a large non-labeled dataset improves the performance of the regression model trained on a smaller dataset.

Id	**StartTime**	**VehicleLength**	**GrossWeight**	**Velocity**
1	12/12/2021 23:59	4.43 m	1570 Kg	125 Km/h
2	12/12/2021 23:59	4.18 m	1625 Kg	113 Km/h
3	12/12/2021 23:57	5.03 m	885 Kg	96 Km/h
...	...	...	...	...
99	12/12/2021 23:48	16.78 m	27780 Kg	77 Km/h
100	12/12/2021 23:48	4.55 m	1400 Kg	120 Km/h

Table 6.1. Example of label data generated by the scale.

6.3 Data preprocessing

6.3.1 Signal preprocessing

Following the process described in section 4.1.2, and similar to the preprocessing in section 5.1.2, the dataset was divided into overlapped windows of length $T_{win} = 60s$ and stride $T_{stride} = 15s$.

The resulting windows had a dimension of (6000,1), and were transformed into PSD spectrograms using a $frame_{Lenght} = 198$ samples and a $step_{Lenght} = 10$ samples. The resulting spectrograms are 2D arrays with a shape of (100, 100). Figure 5.3B shows some examples of the obtained spectrograms.

6.3.2 Labels generation

As mentioned in section 6.2, the labels in this dataset are obtained from a scale located 600m before the first sensor. Due to the distance between the scale and the sensors, and the random nature of vehicle crossings, there is no direct way to link the records in the scale with the perturbances detected by each sensor. To associate the labels with the corresponding disturbances in the sensor signals, a matching procedure was developed. Our pipeline consists of four stages, which are described in the following sections.

Feature extraction and threshold definition

Starting from the assumption that when a vehicle crosses the viaduct the variability in the measures of the sensors increases, our pipeline relies on analyzing the variance in the sensor records. We begin by calculating a moving variance window, Var_{win}, of 1 second for each record. Figure 6.2 shows how Var_{win} amplifies the perturbations in the signal, making them easier to identify and isolate.

To isolate the white noise in Var_{win} and identify the pieces of a signal corresponding to a disturbance, we use the interquartile statistical rule. This involves calculating the first and third quartiles, $Q1$ and $Q3$, and then determining the interquartile range, IQR, as $IQR = Q3 - Q1$. We then calculate the lower and upper bounds as $lwb = Q1 - 1.5 * IQR$ and $upb = Q1 - 1.5 * IQR$, respectively, and discard all records outside these bounds. Once we have isolated Var_{win} to the white noise, we calculate its mean, $miu(Var_{win})$, and standard deviation, $std(Var_{win})$. Finally, we calculate a threshold for each sensor using the following equation: $th = miu(Var_win) + 3.5 * std(Var_win)$. This threshold will be useful in the next section for identifying vehicles.

Vehicles identification

For each sensor, we filter Var_{win} with the previously calculated threshold for each sensor. The goal is to identify groups of consecutive records that exceed the threshold. If a group has a duration greater than a given threshold T_{min}, we consider it a possible vehicle. For each possible vehicle, we extract the following information: the energy of the signal, which we calculate using equation 3.1, and the start and end time of the perturbation.

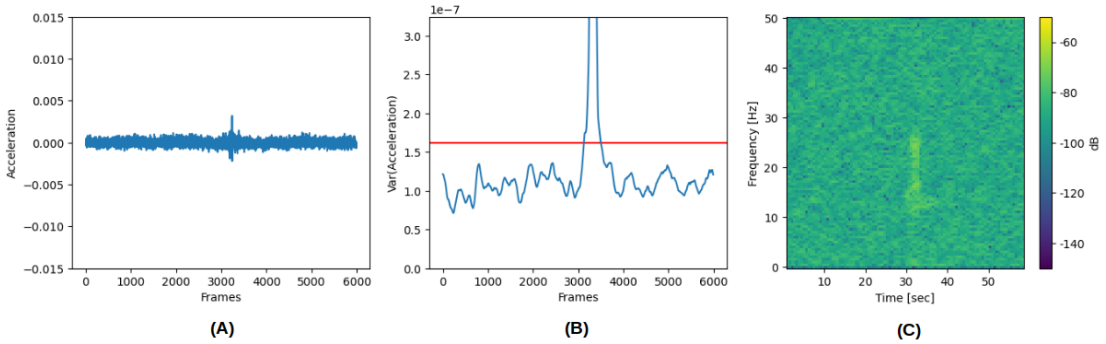

Figure 6.2. Example of a signal, variance amplification plot, and resulting spectrogram.

A) The illustration depicts a signal generated by a vehicle crossing the viaduct. B) shows the corresponding variance plot from the signal. The variance plot is useful because it allows amplifying the disturbance generated by the vehicle making it easier to isolate with a threshold. C) The corresponding signal spectrogram.

Labels preprocessing

To accurately match the labels with the perturbations in the sensors' records, we need to estimate the time at which each registered vehicle will cross each sensor. Since each vehicle detected by the scale has a precision of 60 seconds, we need to extract additional features to estimate the exact time of the crossing. To accomplish this, we extract the following features:

A Minimum time: It is calculated as the timestamp plus the time to go from the scale to the sensor according to the distance between them, and the speed of the vehicle. We subtract 5 seconds to account for cases when the vehicle accelerates on the road.

B Maximum time: It is calculated as the minimum time plus 60 seconds of uncertainty, plus 30 seconds to account for cases when the vehicle decelerates on the road.

By extracting these features, we can estimate the time range during which each vehicle will cross each sensor. This information is crucial for accurately pairing the labels with the corresponding perturbations in the sensors' records.

Labels assignation

The labels are matched with the signal perturbations captured by the sensors through the following procedure. First, the labels are sorted from the heaviest to the lightest. Then, for each label, a list of candidate groups is generated. The candidates are the groups that overlap with the minimum and maximum time extracted from the label's preprocessing stage. Finally, from the candidates, the one with the highest energy is assigned to the label.

6.4 Masked-autoencoder settings

6.4.1 Patch embeddings

The 2D spectrograms are divided into non-overlapped regular grid patches of patch size $P = 5$, generating a total of 400 patches.

6.4.2 Masking

Using a masking rate $Mrate = 0.8$, in total 320 patches are randomly masked, and only 80 patches are visible for the autoencoder.

6.4.3 Autoencoder architecture

During pre-training, the autoencoder is composed of three layers in the encoder and four layers in the decoder. During fine-tuning, the decoder is discarded, and a neural network is attached on top of the encoder. The following architectures for the neural network were tested:

- Linear projection on the cls token: A linear projection layer from the classification token in the encoder, to the regression output. (768, 1).
- One hidden layer on the cls token: A multi-layer perceptron with an input layer of 768 neurons, a hidden layer of 384 neurons, and an output layer of one neuron.
- Linear projection on the encoder output: A linear projection layer from the complete encoder output to the regression output. (61440, 1).

6.4.4 Autoencoder objective

During pre-training the objective of our autoencoder is the MSE 4.2 between the predicted and input spectrogram, averaged over the masked patches. During fine-tuning, the objective is the MSE 4.2 between the predicted number of vehicles contained in the signal window and the ground truth.

Chapter 7

Experimental Results

This section presents the experimental results of the masked autoencoder on the tasks discussed in previous chapters. For the anomaly detection task introduced in chapter 5, first, we determined the optimal architecture for the autoencoder, we compared three different architectures with varying numbers of layers in the encoder and decoder. We then trained each architecture and evaluated their performance in three different areas: reconstruction accuracy, computational resources required during training, and inference time. Based on these evaluations, we selected the architecture that achieved the lowest reconstruction error while also minimizing training and inference time. Subsequently, we proceeded to determine the best aggregation strategy, as discussed in Section 5.3. We experimented with different window sizes and evaluated the accuracy of each aggregation method while also minimizing detection delay. Once we had identified the optimal aggregation strategy, we compared the performance of our best model with the state-of-the-art methodology described in Section 3.1.

For the TLE task introduced in chapter 6, we first pre-trained a base autoencoder architecture. Next, we discarded the decoder and proposed three neural networks to be attached to the encoder during the fine-tuning phase. We selected the architecture that minimized the MSE error in the regression task. Finally, we replicated the state-of-the-art methodology introduced in section 3.2, and compared the performances of our best model with it. Additionally, we studied the impact of the pre-training phase on the final performance of the masked autoencoder.

The experiments were conducted using Python 3.8 and several libraries, including Pandas, Numpy, and Scipy for data preprocessing. The masked autoencoder was implemented using the PyTorch framework and the Pytorch

Image Models (timm) library. All the experiments were performed using a single GPU nVidia Tesla V100 32GB.

7.1 Results on Anomaly Detection

7.1.1 Autoencoder architecture

Choosing the right size for a model is crucial for several reasons. First, it helps to avoid overfitting. If a model has too many parameters relative to the complexity of the task, it may fit the training data too closely, resulting in worse performance on the test data. Second, it helps to conserve computational resources by avoiding the unnecessary use of excessive parameters. Finally, choosing the right size also allows embedding the model on edge devices that typically have limited computational power and storage capacity.

Previous applications of the masked autoencoder on larger, noisier, and more diverse datasets have suggested an optimal size of 12 layers in the encoder and 16 layers in the decoder [19][20]. However, we need to determine the optimal size of the autoencoder for this specific application.

We trained the architectures proposed in Table 5.1 according to the following training details.

Implementation details

As mentioned in Section 5.1.1, the available dataset comprises fifteen days of data, with four days before the intervention and eleven days after it. The dataset was partitioned considering its temporal dependency to avoid incorporating time-correlated data into different partitions.

The training set comprises 500K spectrograms representing normal data obtained from the first four days after the intervention. The validation set consists of 400K spectrograms representing normal data obtained from the following three days after the intervention. Finally, the test set is composed of three days after the intervention representing normal data, and three days before the intervention representing anomaly data. The test set includes a total of 400k spectrograms, with 200k normal spectrograms and 200k anomaly spectrograms.

Each architecture was trained for 200 epochs using the training set as described previously. The weights were initialized using the Xavier uniform initialization method [16], as discussed in section 4.3.1. The MSE between the predicted spectrograms and the original spectrograms was used as the

loss function during the training process. Finally, after 200 epochs, the performance of each architecture was evaluated on the validation set using MSE as the evaluation metric.

7.1.2 Pre-training results

Table 5.1 presents the results of our experiments on three different architectures. The large architecture achieved the best performance, with the lowest reconstruction error of 0.000076. However, it required a significant amount of computational resources, with a training time of 300 hours. and a higher inference time.

Architecture	**Training time (Hours)**	**Inference time (Seconds)**	**Reconstruction error (MSE)**
Small	51	0.16	0.000109
Medium	100	0.69	0.000095
Large	300	2.10	0.000076

Table 7.1. Performance, training time and inference time comparison for the tested architectures.

Although the medium and large architectures had better performance, we decided to continue our analysis with the small architecture. This decision was based on the fact that the small architecture had a similar reconstruction error to the medium architecture, but with a significantly lower training and inference time. Using the small architecture, we can make a fair comparison with state-of-the-art methodologies presented in section 3.1, which are based on classical machine learning models.

7.1.3 Anomaly detection pipeline

As described in Section 5.3, our proposed anomaly detection pipeline comprises two stages. In the first stage, we aggregate the reconstruction errors, while in the second stage, we determine a threshold based on the aggregated error. We propose two methods for aggregating the reconstruction error: aggregation by the number of vehicles and aggregation by time windows. We calculate thresholds for each aggregation type and size using the following equation: $th = \mu + n \times \sigma$, where μ and σ are dependent on the aggregated errors, and n is a hyperparameter that needs to be defined. We tested three

different values for n as follows: $n = 2.5$ with a statistical probability of 0.62% for obtaining false positive errors, $n = 3$ with a statistical probability of 0.13% for obtaining false positive errors, and $n = 3.5$ with a statistical probability of 0.02% for obtaining false positive errors. We then select the aggregation method and threshold that maximize the accuracy of the anomaly detection pipeline.

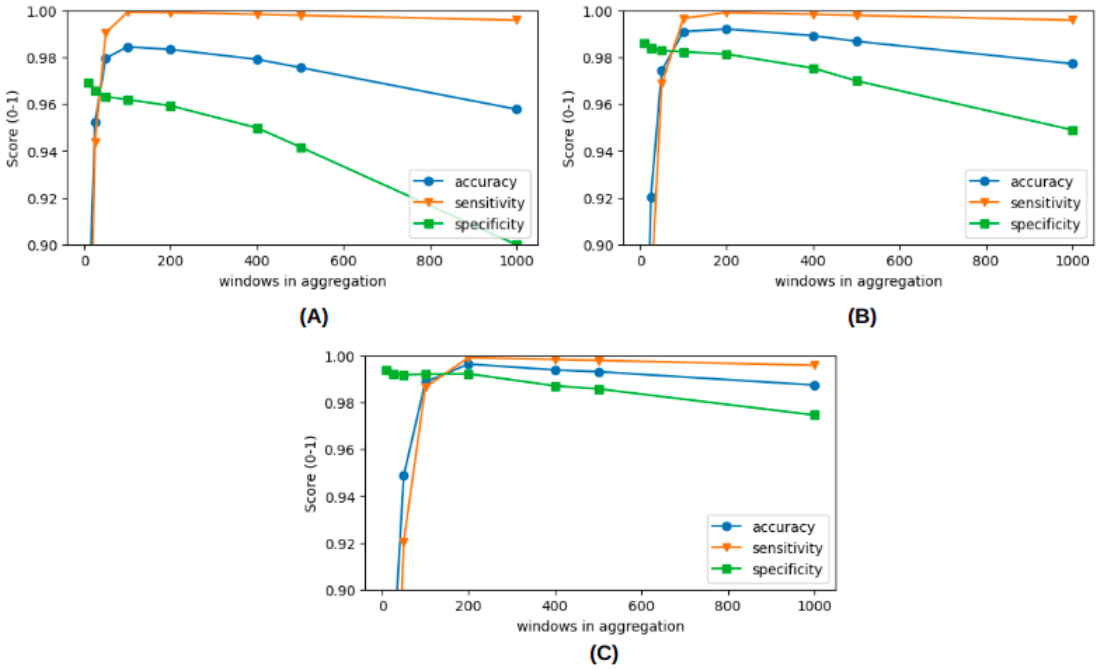

Figure 7.1. Results aggregation by the number of vehicles for each threshold.

The plots correspond to: A) $n = 2.5$, B) $n = 3$, C) $n = 3.5$

- Aggregation by the number of vehicles: We evaluated the performance of our anomaly detection system using aggregation windows of different sizes, ranging from 1 to 500 consecutive reconstruction errors. For each aggregation window size, we calculated three different thresholds using the values of n (2.5, 3, and 3.5).

  Figure 7.1 presents the performance of our system for the different scenarios. Our results show that aggregation windows greater than 200 vehicles achieve a sensitivity of approximately 1, indicating that the system can accurately identify anomaly data for any tested threshold. However, using larger aggregation windows decreases the specificity of

Method	Aggregation size	Threshold (n)	Accuracy	Sensitivity	Specificity
MAE	100 Vehicles	3	**99.4%**	**99.8%**	98.65%
MAE	1 Minute	3.5	98.8%,	99.7%	97.4%
MAE	60 Minutes	3.5	98.7%,	1%	96.8%
PCA [29]	60 Minutes	3	98.8%	97.33%	**100%**

Table 7.2. Comparison between the best-performing models on the anomaly detection task.

the model, making it harder to identify normal windows and increasing the number of false positives detections.

Based on our experiments, the best-performing model was obtained using an aggregation window size of 200 vehicles and a threshold defined as $th = \mu + 3.5 \times \sigma$. This combination achieved an accuracy of 99.6%, with a sensitivity and specificity of 99.9% and 99.2%, respectively.

- Aggregation by time: To compare our results with those of [6], we tested aggregation windows ranging from 1 minute to 4 hours and calculated three different thresholds for each window size, using possible values of n equal to 2.5, 3, and 3.5. Figure 7.2 illustrates the model's performance in different scenarios.

  Overall, the model achieved a sensitivity near or equal to 1 for all cases, indicating its ability to identify anomalies. However, the specificity of the model did not improve as the aggregation window increased in size for any of the possible thresholds. Instead, the specificity decreased, reaching a low of 62% for aggregation windows of 4 hours. Additionally, higher values for the threshold resulted in better performance.

  The best-performing model used an aggregation window length of 60 seconds and a threshold defined as $th = \mu + 3.5 \times \sigma$. This model achieved an accuracy of 98.8%, with a sensitivity of 99.7% and a specificity of 97.4%.

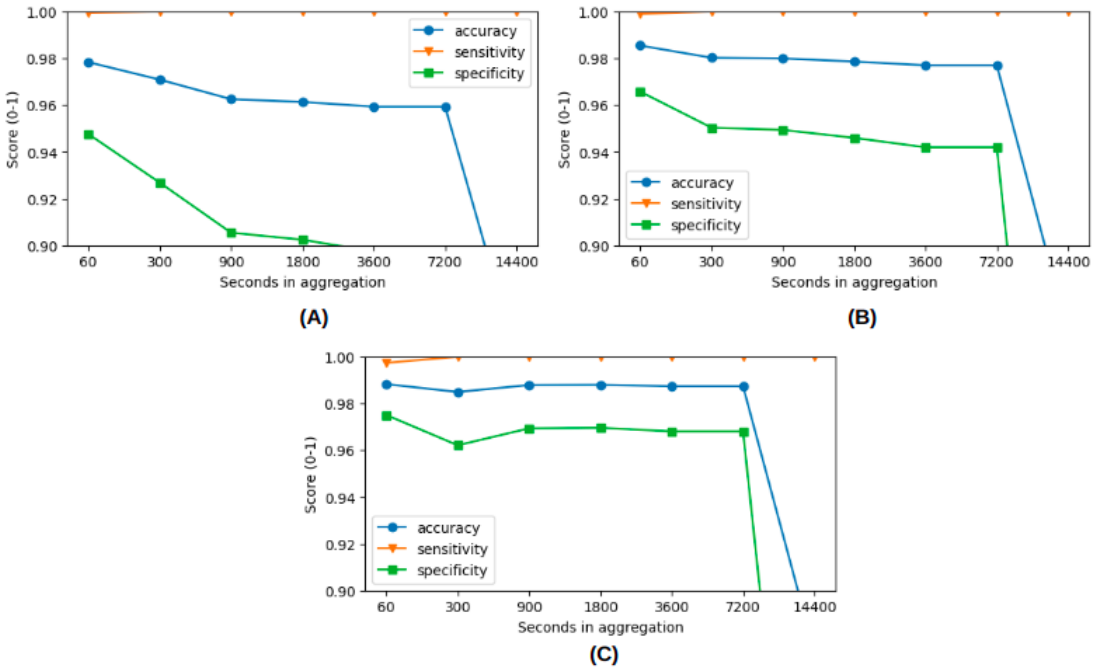

Figure 7.2. Results aggregation by time for each threshold.

The plots correspond to: A) $n = 2.5$, B) $n = 3$, C) $n = 3.5$

Table 7.2 summarizes the information of our best-performing models for each aggregation method, including the results reported on the state-of-the-art methodology introduced in Section 3.1. Our proposed methods outperform the state-of-the-art methodology based on the PCA algorithm in terms of accuracy and sensitivity, demonstrating an increased capability for identifying anomalies. Furthermore, when aggregating by time, our delay in detecting anomalies is only 1 minute, compared to the PCA-based method's delay of 1 hour. This improvement is significant in ensuring the system's reliability in real-time applications.

7.2 Results on traffic load estimation

7.2.1 Label generation

The quality of a model's performance is directly affected by the quality of the dataset. In this application, the dataset's quality depends on the accurate pairing between the scale measurements and the disturbances detected

by the acceleration sensors on the viaduct. As discussed in section 6.3.2, there is no direct way to relate the measurements of both data sources, and the randomness in the behavior of the vehicles and the noise captured by the sensors make it challenging to generate accurate labels for the vibration data. Several system characteristics affect label generation, including the large distance between some sensors and the scale (ranging from 600 meters to 2.5 kilometers), the variable speed of vehicles on the road between the scale and sensors, the time granularity of the scale records, and the natural noise affecting each sensor.

Applying the methodology proposed in section 6.3.2, we were able to pair 54% of the scale records with disturbances detected by each sensor. While this result is reasonable given the characteristics of each dataset, it also means that 46% of disturbances remain present in the sensor measurements but were not properly identified. This makes our dataset noisy and challenging for any machine learning model to learn from.

7.2.2 Pre-training phase

We first trained the masked autoencoder architecture, introduced in section 6.4.3, on a reconstruction task. The training process consisted of 200 epochs on 283k training examples. The objective in this phase was to obtain a model that learned the general dependencies and characteristics of the dataset. The trained model was then further trained in a fine-tuning phase to address the TLE tasks. In the following sections, we introduce the results of the fine-tuning phase and study the impact of the pre-training phase on the model's performance in the TLE task.

7.2.3 Fine-tuning phase

In this phase, the pre-trained autoencoder is modified by discarding the decoder and attaching a neural network on top of the encoder for the regression task. In section 6.4.3, we introduce three different architectures for the neural network to be tested in this phase. Each architecture is trained for 200 epochs on 50K training samples to predict the load of the bridge at a given time. Table 7.4 summarizes the performance of each tested architecture. First, we observed that models using only the output of the encoder's CLS token as input perform better than those using the complete output. The best-performing architecture consists of a linear projection layer from the encoder's CLS token output to the output of the regression task.

Architecture	MSE	MAE	MAPE	R2
Linear projection on the CLS token	**0.343**	**0.393**	**25.58%**	0.53
One hidden layer on the CLS token	0.351	0.408	26.05%	0.53
Linear projection on the encoder output	0.350	0.412	26.26%	0.53

Table 7.3. Performance of the proposed architectures in the fine-tuning phase.

7.2.4 Comparison with state of art techniques

In section 3.2, we introduce [6] where the TLE problem is addressed using Structural Health Monitoring (SHM) data from a viaduct in Italy. Given the similarities with our context, this work serves as a baseline for comparing the performance of our method. However, a direct comparison of our results with the ones reported in [6] would be unfair due to differences in the labeling process. Specifically, on [6] used a 30-minute video recording to semi-automatically generate labels for the data recorded by the sensors, resulting in a high-quality dataset for their proposed methodology.

To make a fair comparison between the two methodologies, we attempted to apply our proposed method to the dataset used in [6]. However, this dataset was not available. Therefore, to compare the two methods, we replicated the methodology proposed in [6], which was introduced in section 3.2. We then used the replicated model on our dataset to have a direct comparison between the methodologies. The process to replicate the methodology was developed as follows.

- Since [6] used only one group of sensors, we selected the group of sensors nearest to the scale for our analysis.
- From the raw vibration signal, we generated a dataset by extracting statistical features for the six sensors composing the selected group, as described in table 3.1. In total, the dataset is composed of 216 features.
- To select the most relevant features for our analysis, we used the correlation-based filter method based on the F-test. The F-test is a statistical test that compares the variance of two groups of data to determine if they are significantly different. We tested 11 different values for the number of

selected features, $n = (5, 10, 15, 20, 25, 50, 75, 100, 150, 200, 216)$, similar to the ones used in [6].

- To train our regression models, we used the optimal hyperparameters reported in [6]. We trained five of the six regression algorithms proposed in [6] using the hyperparameters reported in Table 3.2.

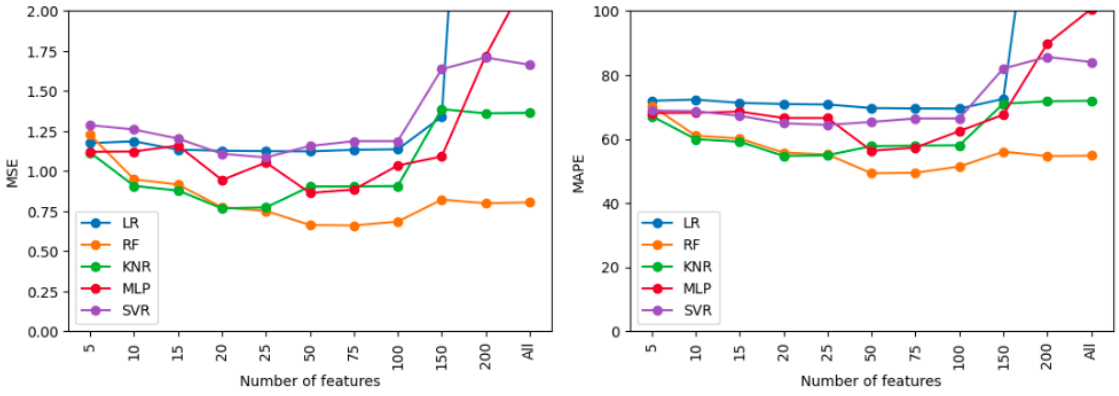

Figure 7.3. Results aggregation by the number of vehicles for each threshold.

Figure 7.3 illustrates the performance of each model trained with each possible number of features, as evaluated by mean squared error (MSE) and mean absolute percentual error (MAPE). In contrast to the results reported in [6], where support vector regression (SVR) was found to be the best algorithm, our results show that the best-performing models are based on random forest (RF) and k-nearest neighbors regression (KNR).

In general, each model achieves the best results when trained with between 25 and 75 features. The best-performing model in our study is an RF model with an MSE of 0.661.

Table 7.4 summarizes the performance of each trained model, as evaluated by Mean Square Error (MSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). Our methodology outperforms the state-of-the-art methods by a wide margin, with an improvement of over 90% in terms of MAE on our dataset, compared to the best-performing state-of-the-art methodology based on RF algorithm.

Architecture	MSE	MAE	MAPE	R2	NumFeatures
MAE	**0.343**	**0.393**	**25.58%**	**0.53**	NA
LR	1.123	0.759	69.68%	-0.01	75
RF	0.661	0.539	49.47%	0.40	75
KNR	0.767	0.596	54.74%	0.30	20
MLP	0.864	0.614	56.34%	0.21	50
SVR	1.085	0.702	64.43%	0.01	25

Table 7.4. Comparison between the performance of the models proposed on [6] and the proposed Masked Autoencoder MAE.

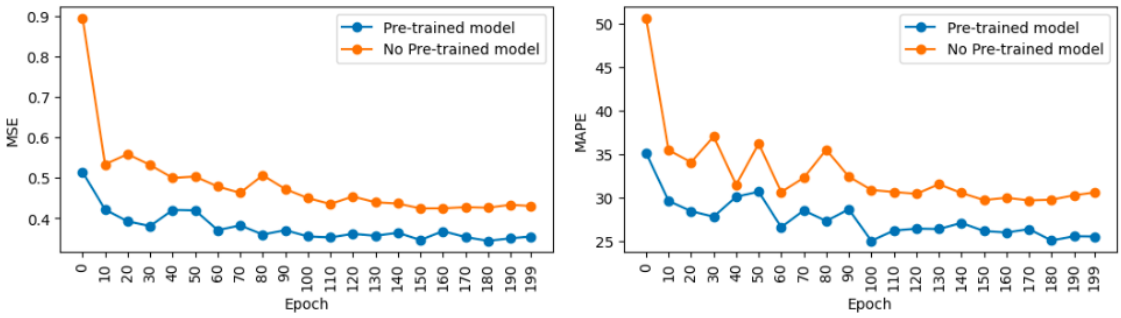

Figure 7.4. Performance Pre-trained Vs No Pre-trained model for an increasing number of training epochs.

7.2.5 Is Pre-training enhancing performance?

On [19][20], it was demonstrated that incorporating a self-supervised pre-training phase can be beneficial to increase the performance of models on downstream tasks such as image classification and audio recognition. In this section, we aim to investigate whether a similar pre-training phase can improve the performance of models on the TLE task.

To investigate the impact of the pre-training phase on model performance, we take the best architecture identified in section 7.2.3 and train this model from scratch without using pre-training. Specifically, we skip the pre-training phase and directly fine-tune the model.

The model was trained for 200 epochs on 50K labeled data points. Figure 7.4 illustrates the variation in the training performance of the model with respect to the number of training epochs. Meanwhile, Figure 7.5 illustrates

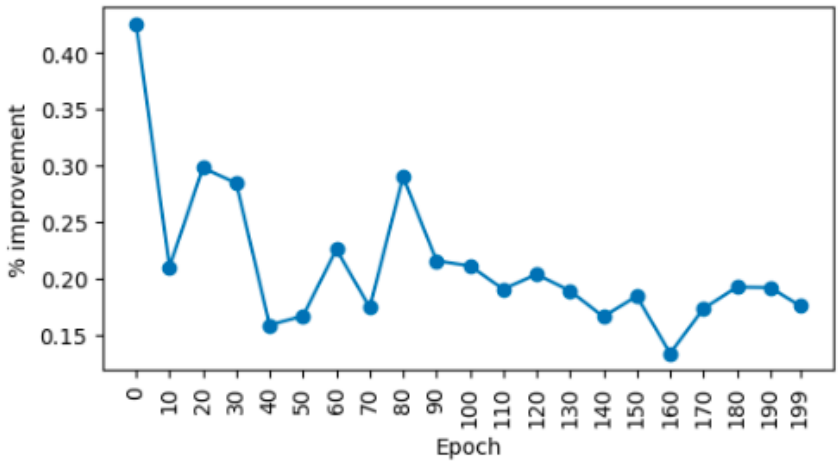

Figure 7.5. Percentual performance improvement of the pre-trained model over the no pre-trained for an increasing number of training epochs.

Architecture	**MSE**	**MAE**	**MAPE**	**R2**
Pre-trained	**0.343**	**0.393**	**25.58%**	**0.53**
No pre-trained	0.429	0.480	30.60%	0.42
% improvement	20%	18.1%	16.4%	20.8%

Table 7.5. Comparison between the best-performing pre-trained Vs no pre-trained models

the percentage difference in the performance of the pre-trained and non-pre-trained models across different numbers of training epochs. In general, the pre-trained model outperforms the non-pre-trained model by 15% to 30% in terms of the chosen performance metric. Specifically, at the end of the 200 training epochs, the pre-trained model performs 17% better than the non-pre-trained model according to the MSE metric.

Table 7.5, summarizes the metrics of the best-performing models for the pre-trained and No pre-trained models. In conclusion, the pre-trained model improves the performance by 20% according to the MSE error.

Chapter 8

Conclusions and Future Work

In this thesis, we propose the use of Masked Autoencoders for SHM tasks. Specifically, we focus on addressing two key tasks in SHM: anomaly detection and traffic load estimation. We achieve this by leveraging self-supervised learning techniques on acceleration data collected from an SHM sensor network

For the anomaly detection task, we trained the autoencoder using a reconstruction approach that exclusively used normal data. We then used the aggregate consecutive reconstruction errors to identify anomaly data by applying a statistical threshold. To optimize the methodology, we tested different configurations of autoencoder architecture, aggregation windows, window size, and statistical threshold parameters. Based on the results, we chose an autoencoder architecture consisting of three transformer layers in the encoder and four in the decoder as the base model. The best-performing configuration was determined to be a window size of 100 vehicles and a statistical threshold defined as $th = mu + 3std$. Our methodology outperformed state-of-the-art methods with an accuracy of 99.4%, compared to the 98.8% accuracy reported in previous research. Moreover, our methodology achieved quicker anomaly detection, with detection times of 1 minute compared to the previous method's detection time of 1 hour, while maintaining the same level of accuracy.

For the traffic load estimation task, we trained the autoencoder in two phases. In the first phase, we pre-trained the autoencoder on a self-supervised reconstruction task to leverage non-labeled data. In the second phase, we fine-tuned the autoencoder by attaching a neural network to the top of the

encoder, and trained it using a labeled dataset to predict the number of vehicles in the viaduct. We tested various architectures for the attached neural network and determined that the best-performing architecture was a linear projection layer attached to the top of the encoder CLS token.

To evaluate our methodology, we compared it with state-of-the-art methods by applying them to our dataset. We found that our methodology significantly outperformed state-of-the-art methods, achieving a MAPE and R-squared equal to 25.58% and 0.53, respectively. In contrast, the best-performing state-of-the-art method based on RF achieved a MAPE and R2 equal to 49.47% and 0.40. Finally, we demonstrated that pre-training the autoencoder using a self-supervised approach with non-labeled data led to a 20% improvement in the model's performance, as measured by the MSE.

Future work includes improving the label generation phase for the TLE task to enhance the quality of the labeled dataset. Another future work is the deployment of the developed methodology on edge devices in the viaduct for real-time traffic monitoring.

Bibliography

[1] Alabadi, M. and Celik, Y. [2020], Anomaly detection for cyber-security based on convolution neural network : A survey, *in* '2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)', pp. 1–14.

[2] Avci, O., Abdeljaber, O., Kiranyaz, S., Boashash, B., Sodano, H. and Inman, D. [2018], Efficiency validation of one dimensional convolutional neural networks for structural damage detection using a shm benchmark data.

[3] Bahdanau, D., Cho, K. and Bengio, Y. [2014], 'Neural machine translation by jointly learning to align and translate'.
URL: *https://arxiv.org/abs/1409.0473*

[4] Bochkovskiy, A. [2020], *YOLOv4: Optimal Speed and Accuracy of Object Detection*, self-published.

[5] Burrello, A., Brunelli, D., Malavisi, M. and Benini, L. [2020], Enhancing structural health monitoring with vehicle identification and tracking, *in* '2020 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)', pp. 1–6.

[6] Burrello, A., Zara, G., Benini, L., Brunelli, D., Macii, E., Poncino, M. and Pagliari, D. J. [2022], 'Traffic load estimation from structural health monitoring sensors using supervised learning', *Sustainable Computing: Informatics and Systems* **35**, 100704.
URL: *https://www.sciencedirect.com/science/article/pii/S2210537922000440*

[7] Chandola, V., Banerjee, A. and Kumar, V. [2009], 'Anomaly detection: A survey', *ACM Comput. Surv.* **41**.

[8] Cheng, Z., Zou, C. and Dong, J. [2019], Outlier detection using isolation forest and local outlier factor, *in* 'Proceedings of the Conference on

Research in Adaptive and Convergent Systems', RACS '19, Association for Computing Machinery, New York, NY, USA, p. 161–168.
URL: *https://doi.org/10.1145/3338840.3355641*

[9] Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. [2018], 'Bert: Pre-training of deep bidirectional transformers for language understanding'.
URL: *https://arxiv.org/abs/1810.04805*

[10] Dong, H., Wang, X., Zhang, C., He, R., Jia, L. and Qin, Y. [2018], 'Improved robust vehicle detection and identification based on single magnetic sensor', *IEEE Access* **6**, 5247–5255.

[11] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N. [2020], 'An image is worth 16x16 words: Transformers for image recognition at scale'.
URL: *https://arxiv.org/abs/2010.11929*

[12] Entezami, A., Sarmadi, H., Behkamal, B. and Mariani, S. [2020], 'Big data analytics and structural health monitoring: A statistical pattern recognition-based approach', *Sensors* **20**(8).
URL: *https://www.mdpi.com/1424-8220/20/8/2328*

[13] Farrar, C. R. and Worden, K. [2007], 'An introduction to structural health monitoring', *Philosophical Transactions of the Royal Society A* **365**, 303–315.

[14] Ghahramani, Z. [2004], *Unsupervised Learning*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 72–112.
URL: *https://doi.org/10.1007/978-3-540-28650-9*$_5$

[15] Ghosh Mondal, T., Jahanshahi, M. R., Wu, R.-T. and Wu, Z. Y. [2020], 'Deep learning-based multi-class damage detection for autonomous post-disaster reconnaissance', *Structural Control and Health Monitoring* **27**(4), e2507. e2507 stc.2507.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/stc.2507*

[16] Glorot, X. and Bengio, Y. [2010], 'Understanding the difficulty of training deep feedforward neural networks', *Journal of Machine Learning Research* **9**(Jun), 249–256.

[17] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. [2014], 'Generative adversarial networks'.
URL: *https://arxiv.org/abs/1406.2661*

[18] Govindarajan, M. and Chandrasekaran, R. [2009], Intrusion detection using k-nearest neighbor, *in* '2009 First International Conference on Advanced Computing', pp. 13–20.

[19] He, K., Chen, X., Xie, S., Li, Y., Dollár, P. and Girshick, R. [2021], 'Masked autoencoders are scalable vision learners'.

[20] Huang, P.-Y., Xu, H., Li, J., Baevski, A., Auli, M., Galuba, W., Metze, F. and Feichtenhofer, C. [2022], 'Masked autoencoders that listen'.
URL: *https://arxiv.org/abs/2207.06405*

[21] Ibrahim, A., Eltawil, A., Na, Y. and El-Tawil, S. [2020], 'A machine learning approach for structural health monitoring using noisy data sets', *IEEE Transactions on Automation Science and Engineering* **17**(2), 900–908.

[22] Kingma, D. P. and Welling, M. [2013], 'Auto-encoding variational bayes', *International Conference on Learning Representations (ICLR)* .

[23] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. [1998], 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324.

[24] Liu, J., Chen, S., Bergés, M., Bielak, J., Garrett, J. H., Kovačević, J. and Noh, H. Y. [2020], 'Diagnosis algorithms for indirect structural health monitoring of a bridge model via dimensionality reduction', *Mechanical Systems and Signal Processing* **136**, 106454.
URL: *https://www.sciencedirect.com/science/article/pii/S0888327019306752*

[25] Loshchilov, I. and Hutter, F. [2019], Decoupled weight decay regularization, *in* 'International Conference on Learning Representations'.

[26] Mao, X.-J., Shen, C. and Yang, Y.-B. [2016], 'Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections'.
URL: *https://arxiv.org/abs/1603.09056*

[27] Martínez-García, M., Zhang, Y., Wan, J. and McGinty, J. [2019], Visually interpretable profile extraction with an autoencoder for health monitoring of industrial systems, *in* '2019 IEEE 4th International Conference on Advanced Robotics and Mechatronics (ICARM)', pp. 649–654.

[28] Mishra, M., Lourenço, P. B. and Ramana, G. [2022], 'Structural health monitoring of civil engineering structures by using the internet of things: A review', *Journal of Building Engineering* **48**, 103954.
URL: *https://www.sciencedirect.com/science/article/pii/S235271022101812X*

[29] Moallemi, A., Burrello, A., Brunelli, D. and Benini, L. [2022], 'Exploring scalable, distributed real-time anomaly detection for bridge health monitoring', *IEEE Internet of Things Journal* **9**(18), 17660–17674.
URL: *https://doi.org/10.1109%2Fjiot.2022.3157532*

[30] Nie, Z., Guo, E., Li, J., Hao, H., Ma, H. and Jiang, H. [2020], 'Bridge condition monitoring using fixed moving principal component analysis', *Structural Control and Health Monitoring* **27**(6), e2535. e2535 STC-19-0114.R2.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1002/stc.2535*

[31] Pu, G., Wang, L., Shen, J. and Dong, F. [2021], 'A hybrid unsupervised clustering-based anomaly detection method', *Tsinghua Science and Technology* **26**(2), 146–153.

[32] Ren, H., Xu, B., Wang, Y., Yi, C., Huang, C., Kou, X., Xing, T., Yang, M., Tong, J. and Zhang, Q. [2019], Time-series anomaly detection service at microsoft, *in* 'Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining', KDD '19, Association for Computing Machinery, New York, NY, USA, p. 3009–3017.
URL: *https://doi.org/10.1145/3292500.3330680*

[33] Rousseeuw, P. J. and Hubert, M. [2018], 'Anomaly detection by robust statistics', *WIREs Data Mining and Knowledge Discovery* **8**(2), e1236.
URL: *https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/widm.1236*

[34] Shalev-Shwartz, S. and Ben-David, S. [2014], *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, Cambridge, UK.

[35] Vanhoeyveld, J., Martens, D. and Peeters, B. [2020], 'Value-added tax fraud detection with scalable anomaly detection techniques', *Applied*

Soft Computing **86**, 105895.
URL: *https://www.sciencedirect.com/science/article/pii/S1568494619306763*

[36] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. [2017], 'Attention is all you need'.
URL: *https://arxiv.org/abs/1706.03762*

[37] Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. [2008], 'Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion', *Journal of Machine Learning Research* **11**, 3371–3408.

[38] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A. [2010], 'Extracting and composing robust features with denoising autoencoders', *Proceedings of the 25th International Conference on Machine Learning* pp. 1096–1103.

[39] Xu, H., Feng, Y., Chen, J., Wang, Z., Qiao, H., Chen, W., Zhao, N., Li, Z., Bu, J., Li, Z., Liu, Y., Zhao, Y. and Pei, D. [2018], Unsupervised anomaly detection via variational auto-encoder for seasonal KPIs in web applications, *in* 'Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18', ACM Press.
URL: *https://doi.org/10.1145%2F3178876.3185996*

[40] Ye, Z., Xiong, H. and Wang, L. [2020], 'Collecting comprehensive traffic information using pavement vibration monitoring data', *Computer-Aided Civil and Infrastructure Engineering* **35**(2), 134–149.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1111/mice.12448*

Acknowledgment: Computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://hpc.polito.it)