CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Attention-based Time Series Forecasting with Limited Data

Attention-based Time Series Forecasting with Limited Data in Electrical Power Systems

Master's thesis in Electrical Engineering

GUSTAV VADSTRÖM

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2024

Master's thesis 2024

Attention-based Time Series Forecasting with Limited Data

Attention-based Time Series Forecasting with Limited Data in Electrical Power Systems

GUSTAV VADSTRÖM

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2024

Attention-based Time Series Forecasting with Limited Data
Attention-based Time Series Forecasting with Limited Data in Electrical Power Systems
GUSTAV VADSTRÖM

Supervisor: Jafar Banar, Electrical Engineering
Advisor: Jakob Lindqvist and Viktor Olsson, Eneryield
Examiner: Paolo Monti, Electrical Engineering

Master's Thesis 2024
Department of Electrical Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Description of the picture on the cover page (if applicable)

Typeset in LaTeX
Gothenburg, Sweden 2024

Attention-based Time Series Forecasting with Limited Data
Attention-based Time Series Forecasting with Limited Data in Electrical Power Systems
GUSTAV VADSTRÖM
Department of Electrical Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Electricity outages are common in electrical power systems, and often caused by natural phenomena, human intervention, or faults in electrical components, such as transformers. A small number of these faults can be predicted by analysing the stream of voltage and current. Forecasting faults in electrical power systems can prevent electricity outages that cause production downtime and capital losses. However, data collected in power systems are usually limited and unbalanced because of the very few historical predictable faults. This study focused on evaluating more recently popular attention-based machine learning models for time series prediction in electrical power systems, in a context where data is a significant limitation. The data was real and consisted of disturbances recorded from power systems over several years, along with documented faults. Two different model architectures were evaluated and compared: the Long short-term memory (LSTM) and the transformer. Three different model instances were trained: using features manually extracted from each disturbance recording, using manually extracted features with pre-training on a similar dataset, and using a signal embedding pipeline attached to each model processing raw waveforms. The results from all six training instances showed that the transformer performed better than the LSTM in terms of evaluation metrics, although the LSTM outputs were more interpretable, because the transformer had higher confidence in its outputs even during false predictions. A bottleneck was found in the small sequence lengths, with improvement shown when utilizing pre-training on a similar dataset containing longer sequences. The integrated waveform feature embedding also showed improvement over the manually extracted features.

Keywords: Computer, science, computer science, engineering, project, thesis, time series, electrical power systems.

Acknowledgements

This thesis concludes my master's degree in Data Science and AI, and I want to thank the people at Eneryield for have given me this incredible opportunity. I want to give a major thanks to my company supervisors Jakob Lindqvist and Viktor Olsson who have been my main advisors from the start of the thesis. Big thanks to my university supervisor Jafar Banar who has given me advise and insight in how I can improve the paper. I would also like to thank my examiner Paolo Monti for wanting to be my examiner for this project. I am grateful for have being a part of a great team of engineers and for have contributed to a tool that can be very useful in the real world.

Gustav Vadström, Gothenburg, 2024-11-23

Contents

List of Figures

List of Tables

1

Introduction

Electrical energy is one of the most important resources to human kind, and it is used daily in commercial industries as well as private homes. Today's society consumes energy for all sorts of things, such as for powering a car or charging a phone. The demand for electrical energy is constantly increasing, and fossil energy sources are replaced by renewable sources [1]. Looking at electrical vehicles, we can see an increase in use of vehicles that run purely on electrical energy by approximately 35 percent from year to year [2]. This increased use of renewable energy contributes to the necessity of effectively being able to distribute and deliver electrical power across long distances. Electrical power systems are networks of electrical components that generate, transport, and distribute electric power. The electrical power used by consumers is powered by a residential or commercial power system that can be decomposed into generators, transmission systems, and distribution systems. All of the components involved in a system make it vulnerable to faults that can lead to downtime in terms of providing electricity, which often becomes expensive for both the provider and the consumer.

Many of the faults in electrical power systems are transient, which are temporary disruptions often caused by things like lightning strikes and birds. These faults are usually not severe and are automatically detected and interrupted with electrical components. Persistent faults, on the other hand, can also occur and often require physical intervention, which could be costly, not only because of downtime, but also because of the reparation and damage to various components of the network. Detecting early anomalies in the power system allows the operators to immediately address the issue and prevent persistent faults. Detecting faults in electrical power systems using a data driven approach is extremely challenging due to the limited amount of data available. This data limitation arises from the infrequency of failures in power systems. Developing an effective solution for this problem could also aid in addressing similar issues in other research areas.

1.1 Problem and research question

The scope of this project is to predict potential faults in electrical power systems in order to prevent persistent faults and serious damage to components. Predictable faults can be found by analyzing the stream of voltage and current from electrical systems to detect anomalies.

The more specific focus of this project is to investigate attention-based machine learning methods to make predictions on limited and unbalanced time-series data. Data in electrical power grids are very limited and unbalanced because of the few number of faults occurring, which shows the importance of evaluating different model architectures for this type of problem. The research from this study can therefore be relevant for other similar forecasting problems where data is limited and unbalanced.

This thesis investigates more recently popular sequential models commonly used in natural language processing (NLP), namely transformers. These model architectures have shown to work very well in natural language processing [3], and the goal in this study is to evaluate the performance of these models on data with the mentioned limitations in a time-series forecasting problem context.

2

Related Work

A small number of studies have previously been made on the topic of fault prediction in electrical power systems with machine learning algorithms. However, the more recent transformer architecture has not been evaluated for time-series prediction problems in electrical power systems where data is significantly limited.

2.1 LSTM Fault Prediction

A recent study was done by Balouji et al. where they used a long short-term memory (LSTM) recurrent neural network (RNN) to predict future faults using real data from electrical power systems [4]. The researchers tried to predict faults within a time-span of a week by sequentially processing electrical disturbance recordings and manually extracting features using domain knowledge from electrical power systems. The solution includes various machine learning techniques and they successfully managed to predict several faults.

2.2 Encoder-decoder for Norwegian Power System

In a master's thesis study made by Høiem (Norwegian University of Life Science), he predicted faults in Norwegian electrical power systems by using an encoder-decoder RNN model [5]. Høiem first trained an LSTM autoencoder with signal denoising, by adding noise to each signal and letting the model correct them. He then froze the trainable parameters of the encoder and attached a classification head in order to train the model for fault prediction. Compared to the study by Balouji, Høiem embedded the signals using an autoencoder, while Balouji manually engineered the feature vectors representing each signal. The dataset in Høiem's study is used to predict many different faults in power systems, such as regular voltage dips. This means that the dataset used in the study by Høiem is not as limited and unbalanced as the one in this thesis, where the focus is to only predict complete electricity outages, and not voltage dips. Because of the limited amount of historical electricity outages, the problem in this thesis becomes much more difficult than the problem addressed by Høiem. The dataset and problem difference also contributes to different prediction horizons.

2.3 Transformer vs LSTM

Another study related to this thesis was made by Paul Bilokon and Yitao Qiu, where they compared the LSTM and Transformer architectures for time-series prediction on financial data [6]. They used real cryptocurrency order data to make predictions about future order prices. The study compared different types of transformer and LSTM variations for three different problem definitions. The best performing problem formulation was mid-price movement prediction, where they tried to use the average k mid-price to predict the next average k mid-price.

The results showed that the LSTM performed better than the transformer [6]. The best LSTM variation obtained an F score of approximately 0.740, while the best performing transformer got an F score of approximately 0.689. This shows that there are instances where the LSTM provides more reliable results in time-series prediction than the transformer.

3

Theory

3.1 Electrical power

Electrical power is the product of current and voltage quantities. The electrical power can have alternating values or constant values, depending on whether it's an alternating current (AC) circuit or direct current (DC) circuit. The advantage of AC power is that it can easily be transformed into different voltages and therefore more easily be transported long distances with minimal energy loss [7]. Alternating current power is periodically reversing its direction and altering its magnitude with respect to time. The AC waveform is most often a sine wave, where the positive period corresponds to moving in a positive direction [8]. The alternating circuit is the most common type of electricity used in electrical power systems, and is therefore what is analysed in this study.

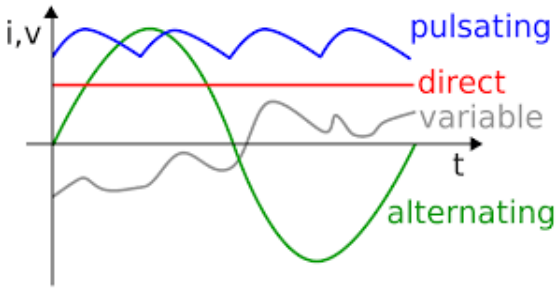

Figure 3.1: Different types of current in electric power. Voltage/current with respect to time. Image from [8].

3.2 Time series forecasting

Time series analysis is the process of analysing historical data in order to make predictions, such as anomaly detection, forecasting, or classification. This process can be applied in several areas, such as in finance, production, or temperature prediction. Predicting the future from historical time series data involves finding patterns and is often done using machine learning algorithms. Depending on the context of the problem, there are many ways to define the mathematical formulation of the forecasting model [9]. For instance, one could create an auto-regressive model

that inputs a number of previous historical data points and output the next. The predicted output can then be used together with the previous history to make further predictions. The auto-regressive formulation is illustrated in equation 3.1, where the function f represents a machine learning algorithm. This model formulation can be useful for problems where the events occurs within a fixed interval from each other and where the time between events is not as important, for instance as in stock market prediction.

$$\begin{aligned} &f(x_1, x_2, ..., x_n) = x_{n+1} \\ &f(x_1, x_2, ..., x_n, f(x_1, x_2, ..., x_n)) = x_{n+2} \end{aligned} \tag{3.1}$$

Time series forecasting can also be applied by predicting the inter-arrival times of events [9]. This means that the model can instead predict the time until arrival of the next event. This model formulation is more time dependent and may suit problems where the inter-arrival time is an important factor. This formulation may also be modified into classification problems, by for instance dividing the output domain into bins. Thus, a time series forecasting model may take historical events as input, and predict the probability of an event occurring on a certain day. Equation 3.2 shows the probabilistic formulation, where d represents the number of days from the point of prediction.

$$P(\text{days}(x_{n+1}) = d \mid x_1, x_2, ..., x_n) \tag{3.2}$$

3.3 Machine Learning

Machine learning is the process of developing an algorithm that learns from data where the statistical challenge is to make the algorithm generalize well such that outputs can be produced for not already seen inputs [10]. Machine learning algorithms generally train by mathematical optimization on data sampled from a real-world distribution. The training dataset should represent the true statistical distribution well in order for the algorithm to generalize over the distribution. There are three general types of machine learning approaches suited for different tasks.

3.3.1 Supervised learning

Supervised machine learning algorithms try to model the relationship between input data and output data [10]. These algorithms are generally developed by mathematical optimization on sampled input and output pairs. The goal of these algorithms is to minimize the difference between the correct labels and the models' outputs. Supervised learning is the machine learning approach that's used in this study.

3.3.2 Unsupervised learning

Unsupervised algorithms are used to find structures and patterns in unlabeled data [10]. This means that it processes data (that does not consist of input and output

pairs) in various ways to find meaningful structures. This could for instance include clustering or finding the distribution of topics among documents.

3.3.3 Reinforcement learning

Reinforcement learning algorithms try to learn a certain task by trial and error [10]. The algorithms generally does not have access to any training data, and therefore has to sample its own data in order obtain a machine learning task that reflects the task solved by the supervised learning algorithms. The data is usually sampled by Monte Carlo techniques that involve engaging with an environment and receiving a reward. The goal of the algorithm is then to take actions within the environment such that the expected reward is maximized.

3.4 Artificial Neural Networks

Artificial neural networks were originally inspired by the biological neural connections in the human brain and is the core essence in the modern deep learning models [11]. State of the art machine learning algorithms consists of a wide range of neural network architectures which consists of neurons interconnected with weights. There are many different types of neural networks, and they can take any form, but the most common structures are feed-forward networks with input and output neurons as well as hidden layers. A feed-forward neural network with one hidden layer is illustrated in figure 3.2.

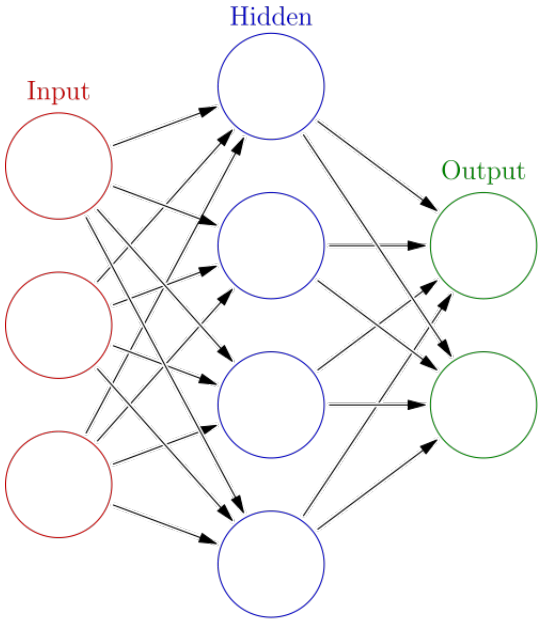

Figure 3.2: A feed-forward neural network with one hidden layer. Takes three values as input and outputs two values. Figure from [12].

3.4.1 Feed-forward and optimization

The process of calculating the values of the neurons in a given layer consists of multiplying the values of the neurons in the previous layer with the coherent weights. This process is then repeated for each of the consecutive layers with corresponding weights. The update formula for a layer L and node i is shown in equation 3.3 [11].

$$V_i^L = g_L \left(\sum_{j=1}^{N_{L-1}} V_j^{L-1} \cdot w_{ji}^L + \theta_i^L \right) \tag{3.3}$$

Here, N_L is the dimension of layer L, w^L are the weights leading into the neurons in layer L, and θ_n^L is the bias term. The activation functions g_L are used to put boundaries on the values of the neurons. The parameters w and θ are then optimized by maximum likelihood estimation or minimization of a loss function through backpropagation. Backpropagation is done by gradient descent on loss function with respect to the parameters. Equation 3.4 shows how the weight parameters in neural networks can be optimized [11].

$$w_{mn}^L = w_{mn}^L - \eta \cdot \frac{d}{dw_{mn}^L} Loss(O(x), t) \tag{3.4}$$

In this update formula, $O(x)$ is the output of the model for input instance x, and t is the target output. The hyperparameter η is the learning rate, and defines the rate at which the network should learn. A too large learning rate may prevent the network from finding a good minimum value, while a too small learning rate can cause the network to get stuck in a local minimum.

A more advanced stochastic optimization algorithm is *Adam* [13], adaptive moment estimation. It's a memory efficient algorithm that uses moment estimation to effectively estimate adaptive learning rates for stochastic gradient descent. The algorithm uses the first and second moment of the gradients to estimate its learning rate. The following equation illustrates how the parameters θ are updated at time step t with gradients g_t embedded in the moment estimates.

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{3.5}$$

where

- α is the step size
- $\hat{m}_t$ is the bias-corrected first moment estimate
- $\hat{v}_t$ is the bias-corrected second moment estimate
- ϵ is a small term to avoid division by zero

3.4.2 Activation functions

The inputs to a neural network are multiplied with the weights of the network to create a linear projection of the input. The projected inputs are then usually passed through a non-linear function in order to make the neural network a universal function approximator [11]. This means that as a result of the non-linearity of the activation function, a neural network can approximate any mathematical function.

Rectified Linear Unit (ReLU) is a common activation function that is linear when $x > 0$ and clamps the negative values to zero, which makes it non-linear [14]. Figure 3.3 shows the function's output for a given input.

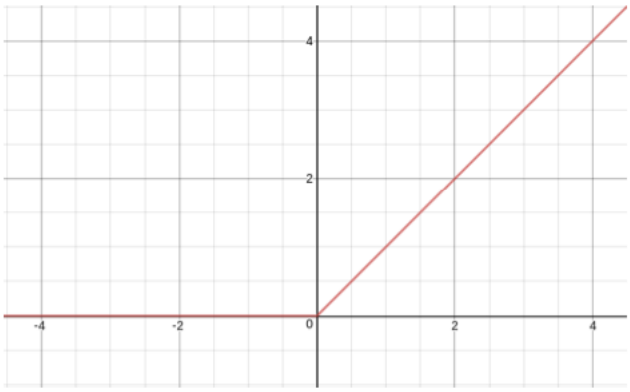

Figure 3.3: Rectified Linear Unit. Figure from [14].

The sigmoid function is another important non-linear activation function that is useful in probabilistic machine learning models [11]. The sigmoid function is defined as in equation 3.6, where the output domain is in $[0, 1]$. This means that the output from a sigmoid function can be interpreted as a probability and is therefore commonly used in the last layer of neural networks for binary classification problems.

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{1 + e^x} \tag{3.6}$$

3.4.3 Loss function

Depending on the machine learning task, there are more or less suitable loss functions. For classification tasks, one of the most common loss functions is the cross-entropy loss [11]. The cross-entropy loss function measures the difference between two probability distributions. The binary cross-entropy log loss is shown in equation 3.7.

$$Loss(O, t) = -(t \log(O) + (1 - t) \log(1 - O)) \tag{3.7}$$

In the binary cross-entropy function, O corresponds to the output probability of the model, while t is the target class (0 or 1). It can be noticed that $log(0)$ is mathematically undefined, and in this case, the term is instead set to a large negative value. In the case of an unbalanced dataset, one might want to penalize incorrect predictions for the minority class. This can by done by introducing a weight that is multiplied with the loss value for a certain class. The formula below illustrates how the loss function can be weighted.

$$Loss(O, t) = w_{t=1} \cdot -(t \log(O) + (1 - t) \log(1 - O)) \tag{3.8}$$

Here, $w_{t=1}$ is dependent on the target value, such that all positive labels are weighted by factor $w_{t=1}$.

3.4.4 Vanishing and exploding gradients

The vanishing and exploding gradient phenomenon is a common problem in deep artificial neural networks [11]. The gradients that are estimated during backpropagation of a neural network can vanish or explode if the network is very deep and consists of many layers. This happens because during optimization of deep neural networks, gradients in the first layers are dependent on the gradients coming from the deeper layers due to the chain rule, which means that the gradients for the more shallow layers are the product of several gradients from the deeper layers. Multiplying many small gradients thus makes the gradients for the shallow layers vanish. If the gradients instead are large, the shallow layers will have large gradients.

The problem with vanishing and exploding gradients is that for deep neural networks, the learnable parameters present in shallow layers will have a hard time obtaining accurate gradient steps [11]. Vanishing gradients will result in no optimization, while exploding gradients results in too large step sizes, also preventing optimization.

3.4.5 Regularization

Overfitting is a common problem during optimization in machine learning algorithms. Overfitting happens when a machine learning algorithm does not generalize well over the target statistical distribution, but instead specifically memorizes the training samples from the distribution. There are several strategies to prevent the algorithm from overfitting, where one of them is utilizing regularization. Regularization is the process of adding a regularization term to the loss function in order to prevent the parameters that we are optimizing to grow too large. This is often done using L2 regularization, which can be included in the binary cross entropy loss function as in equation 3.9 [11].

$$Loss(O, t) = -(t \log(O) + (1 - t) \log(1 - O)) + \lambda \sum_{i=1}^{W} w_i \tag{3.9}$$

Here, the last term is the regularization term, which is the sum over all parameters such that large parameter magnitudes are penalized. Lambda, λ, is the weight decay parameter that controls the amount of regularization to include.

There are more ways to regularize machine learning algorithms, such as introducing noise into the data. This can be done by extending the sampled data with modified data. Augmenting the data broadens the sampled distribution and can help the algorithm with generalization. Noise may also be introduced during training of a neural network with dropout. Dropout is the process of randomly nullifying neurons during training. This provides the model with a stochastic process that cannot be predicted, and thus helps against overfitting.

Another way of reducing the risk of overfitting the data is by implementing early stopping [11]. During optimization, machine learning algorithms are usually trained over several epochs with smaller step sizes, and selecting the appropriate number of epochs can be challenging. Thus, utilizing early stopping helps in preventing the model from over-training.

3.5 Sequential machine learning

Apart from regular supervised machine learning tasks where the model learns to predict outputs given an input, sequential machine learning is the task of inputting sequential data to make predictions. Sequential inputs can therefore have inputs of varying length, and thus requires a different model architecture. Sequential machine learning is applicable in many areas, such as in natural language processing or time-series prediction.

3.5.1 Recurrent neural networks

Recurrent neural networks are a branch of models where the weights are recurrently connected [15]. This means that the model persists a state after each prediction, and thus take previous predictions into account when creating new predictions. Recurrent neural networks are illustrated in figure 3.4, where we can see how predictions are made. During training of the RNN, propagation is done in the same way as for a regular neural network, but backwards through time [16].

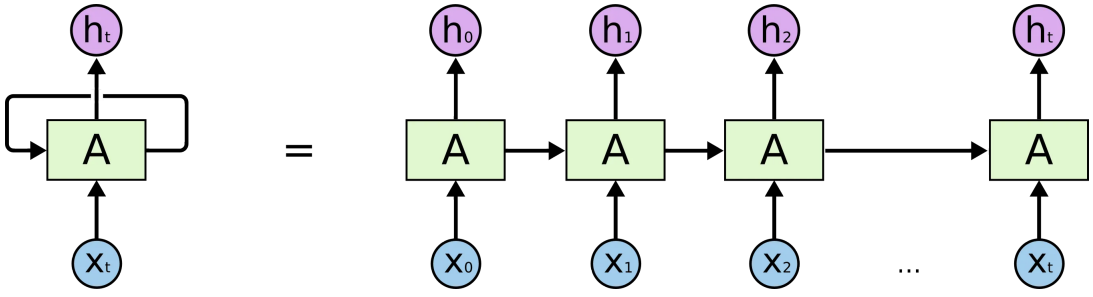

Figure 3.4: Unrolled recurrent neural network. Figure from article [17].

The problem with RNNs is that for long input sequences, the information from earlier inputs are forgotten [16], and the model becomes difficult to optimize due to exploding or vanishing gradients. This is because of the backpropagation through time, and thus the network becomes deeper with each input. This problem can however be somewhat mitigated with the long short-term memory (LSTM) architecture.

3.5.2 Long short-term memory

The Long short-term memory is a recurrent neural network that implements a short-term as well as a long-term memory [18]. The LSTM architecture is illustrated in figure 3.5. The model is structured with the goal of mitigating the vanishing and exploding gradient problem. The hidden state (short-term memory) and long-term memory is usually initialized to vectors of zeros. The hidden state h_t is the output of the model, and is concatenated with the input for each token in the sequence before passed through the model.

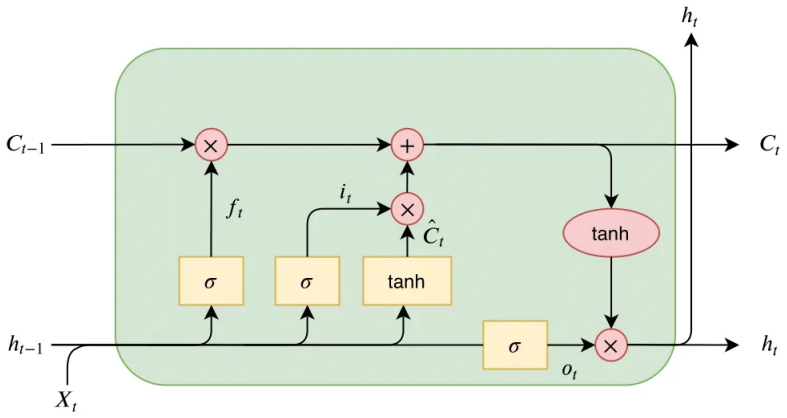

Figure 3.5: The LSTM model. Includes forget gate, input gate, and output gate. Figure from [19].

The model consists of three gates, a forget gate, an input gate, and an output gate. The forget gate passes the concatenated input through a linear layer with a sigmoid activation function. This gives an output value in $[0, 1]$ which is then multiplied with the long-term memory. The vector of scalars represents the portion of long-term memory to keep, and the model will thus forget part of its long-term memory. The forget gate is defined as in equation 3.10, where h_{t-1} is the previous short-term memory, X_t is the current token, and W_f, b_f are the weights and biases.

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \tag{3.10}$$

Unlike the forget gate, the input gate determines what values to add to the long-term memory. The input gate is defined by two operations defined in equation 3.11. In the equation, $\hat{C}_t$ scales between -1 and 1 because of the tanh activation function and outputs what is to be added to the long-term memory C_{t-1}. The outputs from i_t are probabilistic values, which determines the portion of $\hat{C}_t$ to keep before adding it

to the long-term memory. Equation 3.12 illustrates how the new long-term memory is computed.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, X_t] + b_i) \\ \hat{C}_t &= \tanh(W_C \cdot [h_{t-1}, X_t] + b_C) \end{aligned} \tag{3.11}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t \tag{3.12}$$

The output gate determines the new hidden state (the output of the model) by combining the previous hidden state, the input, and the new long-term memory. The output formula is described in equation 3.13

$$\begin{aligned} o_t &= \sigma(W_o \cdot [ht-1, X_t] + b_o) \\ h_t &= o_t \cdot \tanh(C_t) \end{aligned} \tag{3.13}$$

Even though the LSTM architecture somewhat mitigates vanishing gradients for longer sequences, the problem still persists. The more recent attention-based models, such as the transformer architecture, attempts to solve this problem and has been shown to perform well [20].

3.6 Transformer Architecture

The transformer architecture is a sequential neural network that makes use of positional encoding, attention layers, normalization layers, feed-forward layers, and residual connections. The key feature of the transformer is self-attention [20], which prevents backpropagation from happening through time. The self-attention mechanism is what makes the model useful for sequential machine learning tasks, and has shown to perform very well in NLP [3].

The transformer model generally includes an encoder and a decoder used for sequence-to-sequence prediction [21]. However, for a classification task, it's sufficient to use an encoder connected to a classification head. Figure 3.6 represents the structure of a one layer transformer encoder.

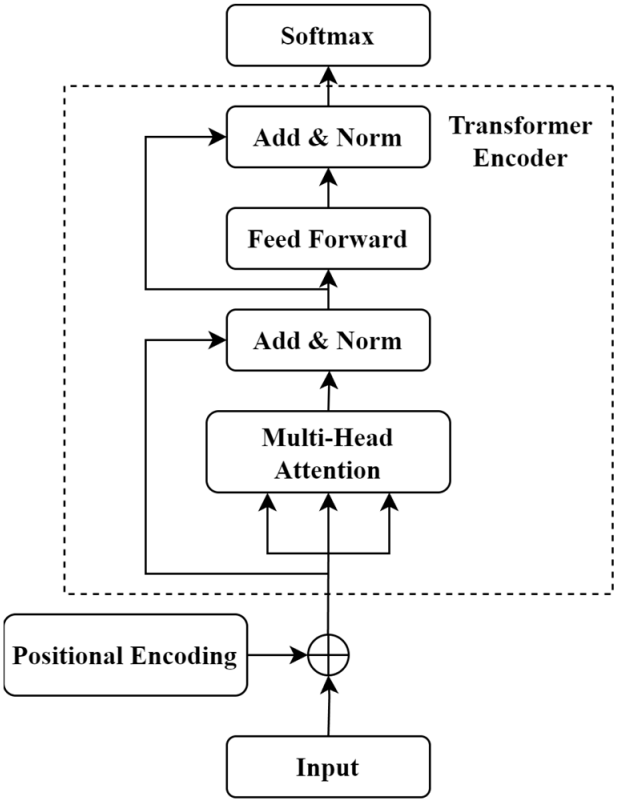

Figure 3.6: Figure from [21].

3.6.1 Positional encoding

Sequential input data are passed through the transformer model by inputting a sequence of fixed length data. One of the main features of the transformer model is that each token of a sequence can be passed through the model simultaneously, which allows for parallel computation [20]. This is partly possible because of the positional encodings, which encodes each token depending on its position in the sequence.

The positional encodings consists of sinus and cosine waves with varying frequencies. The general formula is shown in equation 3.14 [20].

$$P(k,i) = \begin{cases} \sin(\frac{k}{n^{2i/d}}), & \text{if } i \text{ is even} \\ \cos(\frac{k}{n^{2i/d}}), & \text{if } i \text{ is odd} \end{cases} \tag{3.14}$$

Here, $P(k,i)$ is the positional encoding matrix, where k is the position in the sequence and i is the index in the d-dimensional data. The scalar n is self-defined and usually set to a large value. The positional values are then added to the input data, such that the model can recognize the positional pattern and thus pay attention to the structure of the sequence.

3.6.2 Attention mechanism

The self-attention mechanism calculates attention scores between each pair of components in the input sequence [20]. The self-attention is part of the embedding of the input, and combined with the positional encoding, makes each embedded component independent from the other. This means that once the positional encodings and attention scores has been added to the input, it can be independently passed through the rest of the deep neural network.

The self-attention module consists of three weight matrices, the query matrix (Q_w), the key matrix (K_w), and the value matrix (V_w) [20]. These are used to create linear projections of the inputs. The query weights are used to calculate a query point for the active input, which is then multiplied with the key points of all other inputs in the sequence obtained by the projection of the key matrix. The output from this procedure is then passed through a softmax function, and thus represents similarity scores between each of the inputs in the sequence. The value matrix provides value scores for each of the inputs, which are multiplied by the similarity scores in order to obtain the final output of the attention layer. By letting the model learn separate projections for the query, key, and value encodings, it can effectively determine the relevant parts of the input and combine it with the input features. Equation 3.15 represents the self-attention mechanism for a single input $\hat{x}$.

$$softmax\left(\frac{\hat{x}Q_w \cdot XK_w^T + M}{\sqrt{d_k}}\right) \cdot XV_w \tag{3.15}$$

In this formula, X is the input sequence, d_k is the dimension of the keys, and M is the attention mask. The attention mask is used to remove the similarity scores of the padding tokens. For efficient GPU computations, transformer are implemented such that it requires a fixed sequence length as input, and therefore the shorter sequences are padded with dummy tokens. These tokens are then ignored in the attention mechanism by adding large negative values in M for the positions of padding tokens.

3.6.3 Multi-head attention

Self-attention can be taken one step further, by introducing multiple attention-heads [20]. This is done by further project the query, key and value embeddings for each attention-head [20]. Equations 3.16 and 3.17 illustrates the core concept of multi-head attention.

$$\text{Multi-Head}(Q, K, V) = Concat(h_1, ..., h_n) \cdot W^O \tag{3.16}$$

$$h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{3.17}$$

Each head is defined as the self-attention on the projected Q, K, and V [20]. This introduces additional parameter matrices, however, multi-head attention is often

implemented in such a way that the Q, K and V matrices are split among the heads, which result in the same time complexity as for the regular single-headed self-attention.

Multi-head attention has shown to be beneficial as it allows each head to capture different information from different subspaces of the input feature vector [20].

3.7 Convolutional neural networks

Convolution is a mathematical operation that includes integrating over two functions such that it produces a new function [15]. In discrete mathematics, the convolution operation results in sliding a number of discrete values across other values. For instance, sliding a filter across an image. The discrete convolution operation is defined as in equation 3.18.

$$(f * g)[n] = \sum_{k=-\inf}^{k=\inf} f(k)g(n-k) \tag{3.18}$$

The idea of including the convolution operation in neural networks is to let the neural network optimize the filters (the g functions in the equation above) applied to the input in order to effectively extract features [15]. Convolutional neural networks (CNNs) consists of a number of kernels (filters consisting of weights) that is convolved with the input data. Each kernel produces a feature map that represent the input filtered with the specific kernel. This allows for each kernel to extract unique features from the input data. The CNN often also includes a number of pooling layers. Pooling layers are used to reduce the dimension of the feature maps. This is ususally done by convolving the feature maps with max or averaging kernels. The CNN outputs feature maps, which are usually flattened and passed further into a regular feed-forward layer. Image 3.7 illustrates how an input signal can be processed by a convolutional neural network. The output from the CNN is a single vector with extracted features that should represent the input signal well.

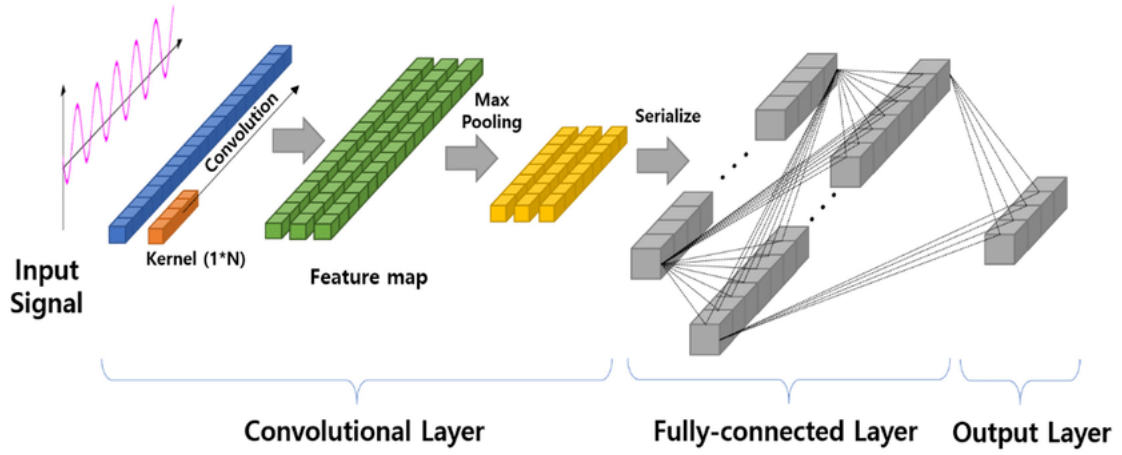

Figure 3.7: A one-dimensional convolutional neural network for signal processing. Image from [22].

3.8 Evaluation

There are several evaluation methods that may be utilized when evaluating how well a data driven algorithm work. The most common way of evaluating a machine learning model is by splitting the sampled dataset into a training and evaluation dataset, where the training set commonly includes the majority of data. The model is then trained on the training data and later evaluated on the evaluation data. The evaluation data is therefore a portion of the data that the model has not already seen, and therefore this step can provide insight in how well the model would perform in production.

For evaluation of classification models, there are several evaluation metrics available to help understand the true performance, such as accuracy, specificity, or recall scores [23]. However, purely looking at a single metric, such as accuracy, can be misleading in terms of unbalanced datasets. A classifier that only predicts the majority class on unbalanced data may receive a high accuracy score even though the classifier does not properly work.

3.8.1 F-score

F-score is the harmonic mean between the precision and recall scores [23]. Both the precision score and the recall score are based on the number of true, false, negative, and positive classifications, which can be illustrated in a confusion matrix, see figure 3.8. The precision and recall scores are evaluated as in equation 3.19.

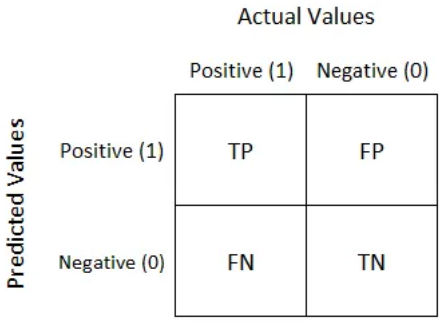

Figure 3.8: Confusion matrix, image from [24].

$$\text{precision} = \frac{TP}{TP + FP} \qquad \text{recall} = \frac{TP}{TP + FN} \tag{3.19}$$

As can be seen in the formula, precision score penalizes the number of false positives predicted, and the recall score penalizes the number of false negatives predicted. This means that the F-score will provide a good metric for how well the machine learning model performs independent of the number of samples and whether the dataset is unbalanced or not.

3.8.2 Specificity

The specificity evaluation score is similar to precision, but instead measures the fraction of true negative predictions. Equation 3.20 illustrates the formula [23].

$$\text{specificity} = \frac{TN}{TN + FP} \tag{3.20}$$

3.8.3 Cross-validation

Cross-validation is an evaluation technique where the data is split into several equal portions (usually five). The machine learning model is then trained on four out of five parts of the data and evaluated on the last part [25]. This procedure is repeated over all five combinations and the final evaluation score is then the average score over all sessions.

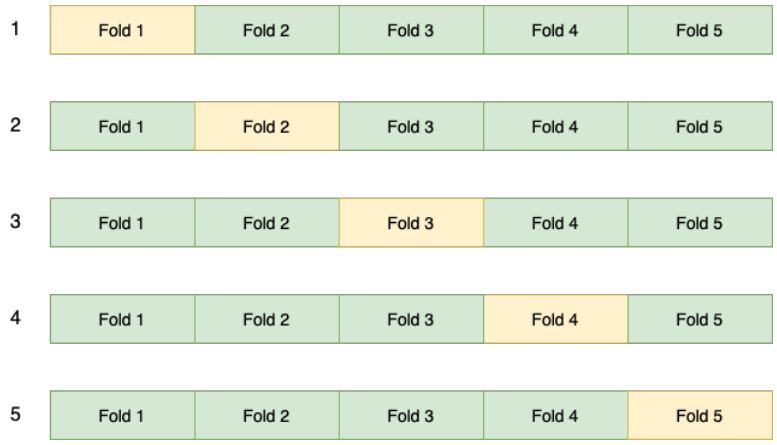

Figure 3.9: Cross-validation with five folds. Green illustrates the data used for training and yellow is the data used for evaluation. The experiment is repeated for five iterations with different evaluation portions.

3.9 Hyperparameter tuning

Hyperparameters are parameters used in machine learning algorithms to specify certain options. Different hyperparameter values might improve or worsen the application of an algorithm on a dataset [11]. These parameters needs to be manually adjusted in order to maximize the performance of the algorithm. It can be any type of parameter, but in the case of artificial neural networks, some of the active hyperparameters are: learning rate, weight decay, and dropout rate. There are generally no efficient method to directly optimize the parameters, and one usually has to rely on exhaustive searches [25].

Another aspect that can drastically affect the performance of an algorithm is initialization of the learnable parameters [11]. The weights and biases in neural networks may be initialized in several different ways, but the values are usually drawn from a distribution. These could be uniform or normal distribution, but there are also distributions that are dependent on the layers and the number of neurons.

3.9.1 Transfer learning

Fitting a model to a smaller dataset can be a challenge, therefore, looking at similar datasets and already trained models may be a solution. Transfer learning can be used in order to take advantage of an already trained model, and fine-tune the model to fit a similar distribution [26]. There are several applications of transfer learning and many different ways to utilize it.

Transfer learning can also be applied by pre-training a machine learning model on a similar dataset, and then fine-tuning on the target dataset [26]. This means that pre-training on similar data may result in good weight initialization for the fine-tuning, such that the neural network already have weights that extracts important features from the data. Thus, the pre-training will provide a good starting point for

the fine-tuning of the network, and may therefore give better result compared to no pre-training. Transfer learning can also be applied by freezing the first few layers of the neural network after pre-training. This would let the network keep the same weights that has learned the feature extraction from the pre-training dataset, while the last layers will be optimized for the fine-tuning dataset. Thus, after fine-tuning, the network will keep information from both datasets.

3.10 Signal processing

Signal processing is the procedure of transforming signal data into a new form allowing us to view properties of the data that could not be viewed before. Signal data can be defined as values changing over some dimension where the data points close to each other are related. This could for instance be audio data, image data, or any other waveform data, such as voltage and current streams.

3.10.1 Fourier transform

The Fourier transform is a mathematical integral transform that operates on a function [27]. The Fourier transform outputs a function that represents the present frequencies and their amplitudes in the original input function. Equation 3.21 shows the definition of the Fourier integral transform, where $f(x)$ is the original signal and $F(k)$ are the function in frequency domain. It should be noted that the Fourier transform works with complex-values functions, and the amplitudes can thus be illustrated by taking the magnitude of the output function $F(k)$.

$$F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} dx \tag{3.21}$$

3.10.2 Continuous wavelet transform

The continuous wavelet transform is closely related to the Fourier transform, but transforms a function into a time-frequency domain in order to get a representation of what frequencies are present at what times [28]. The operation involves a convolution between the original signal and a wavelet function with different scale and translation parameters. Depending on the different scale and translation parameters of the wavelet function, the wavelet transform makes a trade-off of accuracy between the times and frequencies. The wavelet transform is defined as in equation 3.22, where $x(t)$ is the original signal and $\psi(t)$ is the wavelet function. The scale and transition parameters are represented by a and b respectively.

$$X_w(a,b) = \frac{1}{|a|^{1/2}} \int_{-\infty}^{\infty} x(t) \psi(\frac{t-b}{a}) dt \tag{3.22}$$

3.11 Principal component analysis

Principal component analysis (PCA) is a dimensionality reduction technique that is used to reduce the dimensionality of data by selecting principal components such that the variance between each data point is maximized [29]. This means that the features of the data will be modified such that most of the information will be in the first principal components and then subside. The principal components are found by ranking the eigenvectors and eigenvalues of the covariance matrix such that the vectors with the highest eigenvalues are selected. Figure 3.10 illustrates how two dimensional data can be reduced to one dimension using PCA.

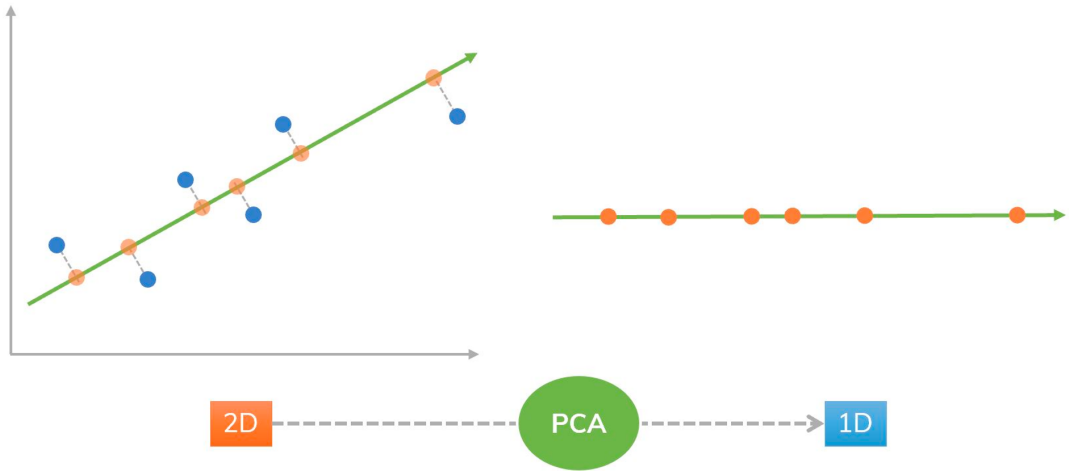

Figure 3.10: 2D data reduced to 1D with PCA. Image from [30].

4

Methods

The workflow during this research was done by incremental improvements and pipeline optimization in order to obtain the best possible results. Several methods and approaches were investigated, such as data augmentation, as well as experimenting with different model architectures and comparing them to baselines.

4.1 Problem modelling

As previously described, the problem is to predict potential future faults in electrical power systems given sequences of disturbances recorded from time-series voltage and current data. The problem can be defined and modelled in several different ways. In this study, I focused on modelling the problem as a binary classification problem where the objective is to predict the probability of a future fault given previous disturbances. The probabilistic model can be defined as

$$P(y \mid x_1, x_2, x_3, .., x_n). \tag{4.1}$$

The model describes the probability of an electricity interruption y happening, given a sequence of disturbance recordings $\mathbf{x}$. In this definition, y is a binary label indicating whether an interruption occurs within a number of days from the point of prediction. There are many more ways of modelling this problem, such as modelling the time until interruption from the point of prediction [9]. However, since the dataset used in this study is very limited, and because there are very few recorded interruptions, modelling the problem as binary was seen as beneficial. This way of modelling the problem creates a time series classification problem instead of time series forecasting, which, in this case, is a relevant restriction. Structuring the problem as a forecasting problem would include either using a continuous output value (regression) or dividing the time-until-interruption space into several subspaces, creating multiple prediction classes. Thus, it is trivially understood that the problem would require more data if structured as a forecasting problem.

4.2 Dataset

The primary dataset, dataset A, used in this study is real-world data collected by gauges in electrical power systems and comes from eight different locations. The data consist of multiple disturbance recordings, each of which is a recording of current and voltage at times when an anomaly occurs. In total, the data spans over several years, and includes a total of 497 disturbance recordings across all locations. The data also includes electrical faults, which indicate times when the electricity supply has been strangled.

The disturbance recordings are approximately one second long and consists of around 2000 measured points of three-phase voltage and current, see figure 4.1. This means that the raw recordings are very high-dimensional and therefore require feature extraction. However, the dataset provided by Eneryield has already been processed such that specific features have been extracted from the voltage and current waves. There are a total of 370 features extracted from each recording, which includes features such as initial/final amplitudes, RMS, and frequency amplitudes for each of the three phases for both the voltage and the current.

Although dataset A was the main source of data used in this study, I also had access to a similar dataset of processed scalars from another location, dataset B. Dataset B is slightly larger, and was utilized as a pre-training dataset for transfer learning.

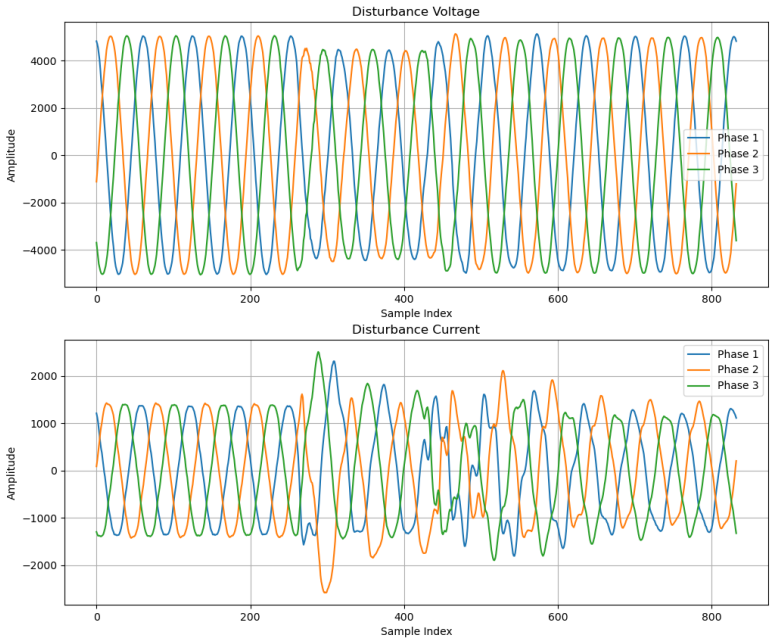

Figure 4.1: A disturbance recording of voltage and current waves for each phase.

4.2.1 Sequential pre-processing

Before the data could be used for time series prediction, further processing of the data was necessary. This includes creating sequential data from the disturbance recordings and interruptions. The sequences were processed by using a specific time-window, which decides how many past recordings from the point of prediction to include in a sequence. This is illustrated in figure 4.2, where we can see that the recordings included in the time-window are also included in the sequence. An additional relative time feature was appended to the front of the feature vector for each recording. This was done to help the model decide the relevancy of each of the recordings, such that the latest recordings are more relevant than the past recordings. This feature was relative and scaled between zero and one.

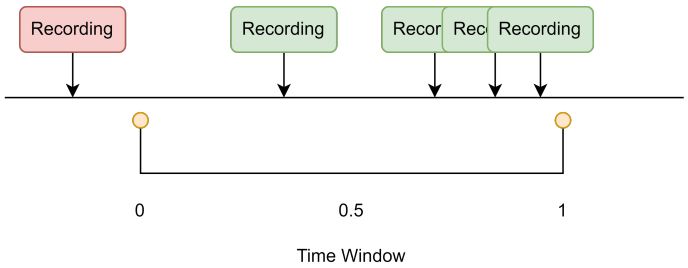

Figure 4.2: The time-line with recordings and a specified time-window. In this example, the green recordings are included in the sequence with respect to the point of prediction (the right-most point of the time-window).

The next step of the sequence processing was to select appropriate times of prediction, or appropriate locations of the time-window. This was done by letting each recording persist at both ends of the time-window. This is almost equivalent to sliding the window across the time-line such that one recording exits or enters the time-window each time. See figure 4.3. By letting each of the recordings persist at the start and end of the time-window, I effectively created two sequences per recording, and thus increased the number of data points. This procedure also helps in keeping variation in the ordering of the recordings in sequences with respect to the time feature. This sequential processing approach was important in order to address the issues of a small dataset, as well as creating more data points with a positive class to tackle the unbalanced data problem.

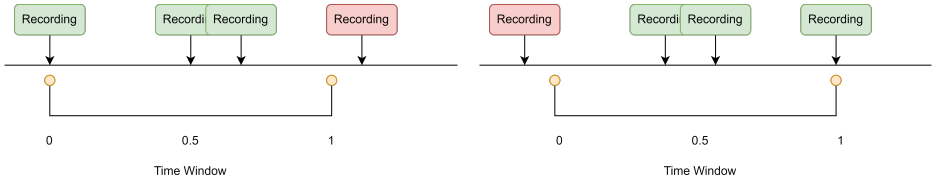

Figure 4.3: The process of sliding the time-window across recordings on the time-line in order to obtain sequences.

4.2.2 Binary labeling

The labeling of the sequences was made by looking at the presence of interruptions a specific time ahead of the point of prediction. If an interruption is within the time span from the point of prediction, the sequence was labeled as positive, otherwise the sequence was labeled as negative. Figure 4.4 illustrates the complete sequencing and labeling process. The time point in between each of the windows is the time of prediction. In this study, it was decided to let both of the windows have a time frame of seven days. This decision was made because it had previously been shown in a study by Eneryield [4], that faults in electrical power systems can be predicted approximately seven days in advance.

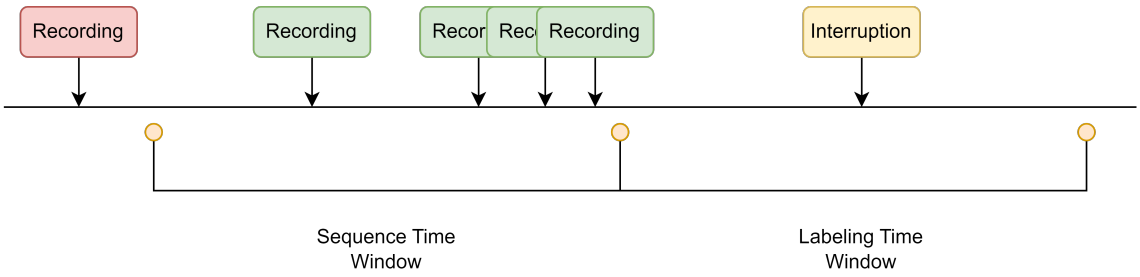

Figure 4.4: The sequencing and labeling process with two different time-windows, one for labeling and one for sequencing.

4.2.3 Data augmentation

In order to reduce the chances of overfitting, the manually extracted features were augmented such that the features for each phase were swapped. Having manually extracted features limits the ways of how the vectors can be augmented, and thus almost only allows for phase rotation. However, using the raw signals for inference allows for a wider range of augmentation options.

The augmentation applied to the raw waveforms included scaling and noise. Both of the augmentation methods were applied to each of the training signals and then concatenated to the original training data. This effectively doubles the amount of sequences for training.

4.3 Model Architecture

Several model architectures were implemented, evaluated and compared for time series prediction of faults in electrical power systems. This includes a trivial baseline solution, a non-sequential feed-forward neural network, a non-attention based LSTM, and a transformer model. Comparing the results of these model architectures for this specific problem would therefore yield a good understanding of their performances on small and unbalanced datasets. The LSTM and transformer models were also implemented and evaluated without relying on the manually processed

features using a specific CNN architecture attached to each of them.

The baseline model is a solution that's used as a point of reference when compared to the more advanced and improved solutions. The baseline model may be a previous machine learning model that we would like to improve upon, but is often a very simple solution that can be used as a sanity check to know that the machine learning models improve over the most basic solution. The feed-forward neural network can be seen as another baseline solution. This model was used to determine the improvement of optimizing using sequences of disturbance recordings instead of single disturbance recordings. The long short-term memory model is used to provide insight in the performance of attention-based algorithms over regular non-attention based sequential machine learning models, in this case, a recurrent neural network. The LSTM and transformer architectures extended with a CNN architecture were used to compare the models in a more generally setting independent of the custom feature vectors. The architectures with a CNN extension thus directly analysed the raw disturbance recordings consisting of voltage and current wavesforms.

4.3.1 Baseline and functional model

The simplest baseline model was a trivial solution that tried to find a correlation between the number of disturbance recordings and interruptions. It was implemented such that it tried to find the number of consecutive disturbance recordings that led to an interruption. The goal was to find an optimal boundary such that all sequences longer than the boundary were classified as positive and the rest as negative. The boundary that produced a minimal binary cross entropy loss were selected as the optimum. However, to prevent the model from predicting all sequences as negative and obtaining a minimum that way, regularization that penalized the boundary length was introduced. The loss with regularization is shown in equation 4.2, where λ is the regularization multiplier while B is the boundary.

$$Loss(O, t) = -(t \log(O) + (1 - t) \log(1 - O)) + \lambda B \tag{4.2}$$

The feed-forward neural network was a functional machine learning model that did not require the data to be sequentially processed and simply took one disturbance recording as input at a time and attempted to predict the probability of a future interruption. The network consisted of three linear layers with a single output neuron passed through a sigmoid activation function. The two hidden layers used ReLU activation function. Figure 4.5 illustrates the functional feed-forward neural network.

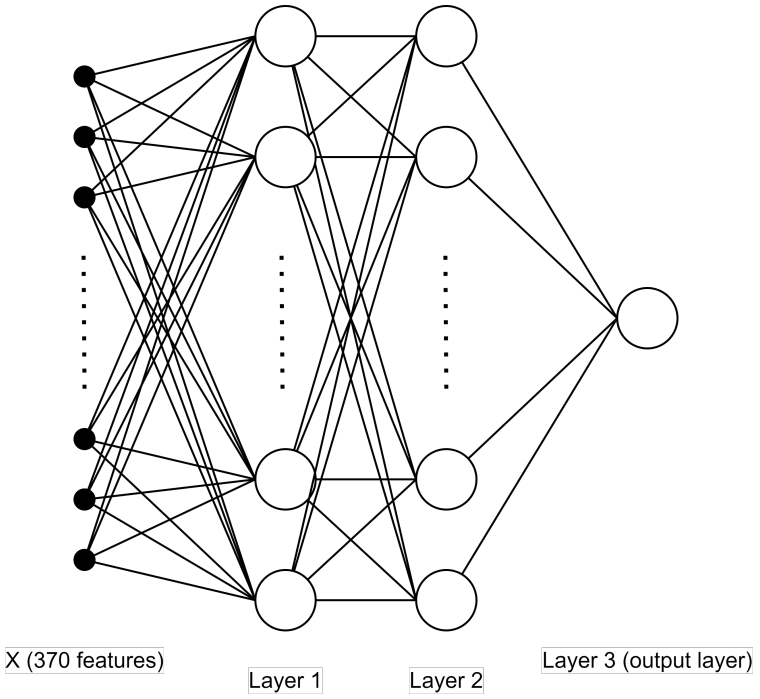

Figure 4.5: The feed-forward neural network. It takes one processed disturbance recording at a time and outputs the probability of a future interruption. The hidden layers use ReLU activation functions.

4.3.2 Long short-term memory model

The long short-term memory was the non-attention based sequential model used to compare with the attention based transformer model [15]. The LSTM model consists of one or more LSTM layers connected to a classification head that outputs a probability. Figure 4.6 illustrates how the model sequentially processes the input data as well as how the classification head is connected to the LSTM architecture. As seen in the figure, the classification head takes the last output from the recurrent layer (LSTM) and passes it through a small feed-forward network of three layers. The last layer in the feed-forward network consists of one neuron with sigmoid activation function. The classification head and sigmoid function thus allows the model to output probabilistic values, which represent the probability of a future fault occurring. Since the LSTM unit is a recurrent neural network, it allows for processing of sequential data, and in this case, sequences of disturbance recordings.

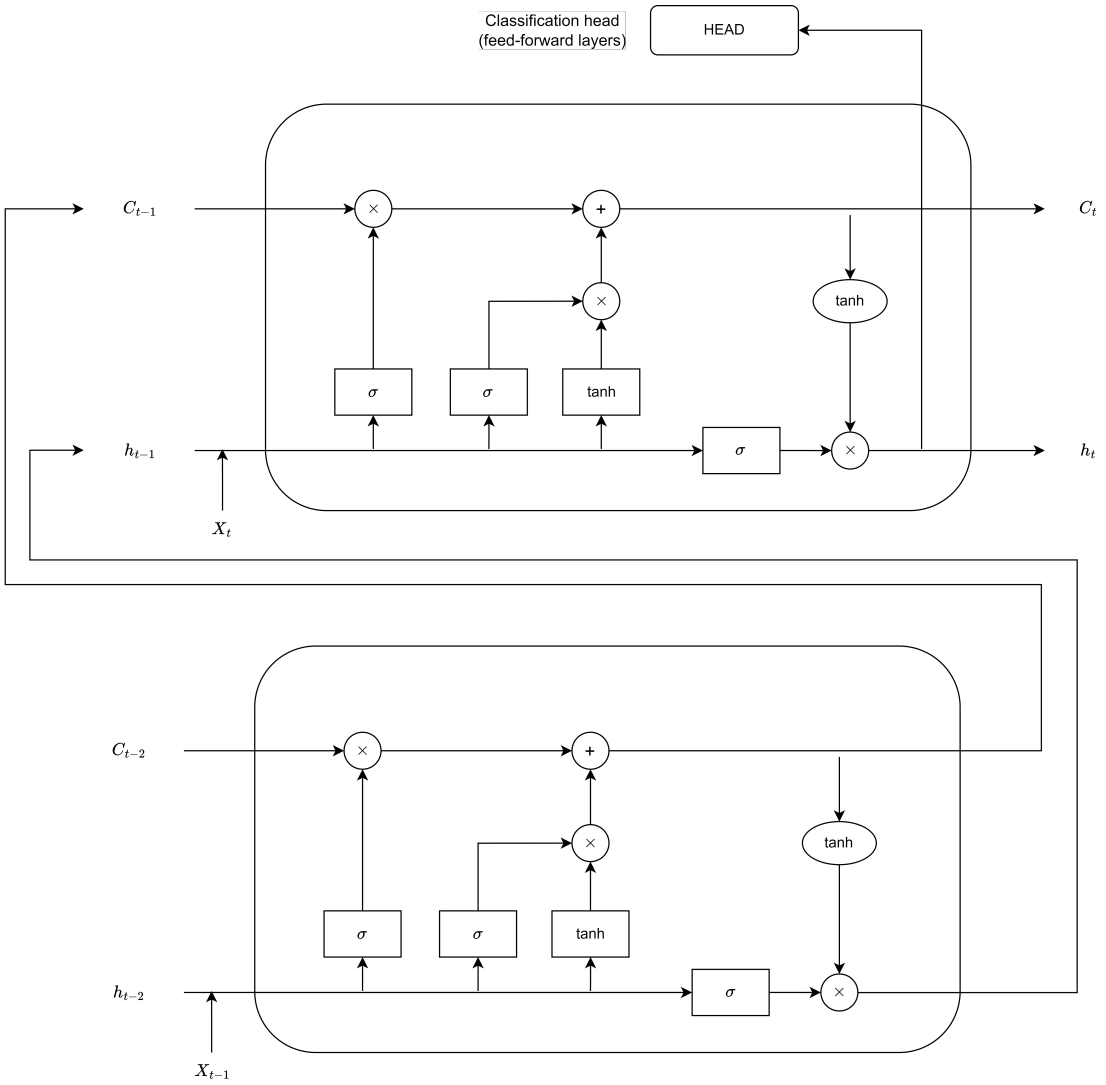

Figure 4.6: The LSTM architecture connected to a classification head consisting of a feed-forward neural network with one output neuron. Here, X_t represents the disturbance recording t in sequence X.

4.3.3 Transformer

The main model architecture used for time-series forecasting on the processed data was a transformer encoder [21], which is illustrated in figure 4.7. The model consists of a positional encoding block followed by the transformer block and a classification head. The model takes a sequence of feature vectors (disturbance recordings) and passes each of them through the positional encoding and transformer block. The transformer encoder outputs one feature vector for each input vector, but only the last output vector is passed to the classification head for prediction. The classification head has the same structure as for the LSTM model, and thus outputs the probability of a future interruption. The transformer model allows for input sequences of variable length, which is important since sequences of disturbance recordings can be of any length [20]. Another important aspect of the transformer architecture is

that it is more reliable when it comes to longer sequences. This is because the transformer is not backpropagated through time in the same way as a recurrent neural network, which makes it less vulnerable to the vanishing gradient problem for long sequences.

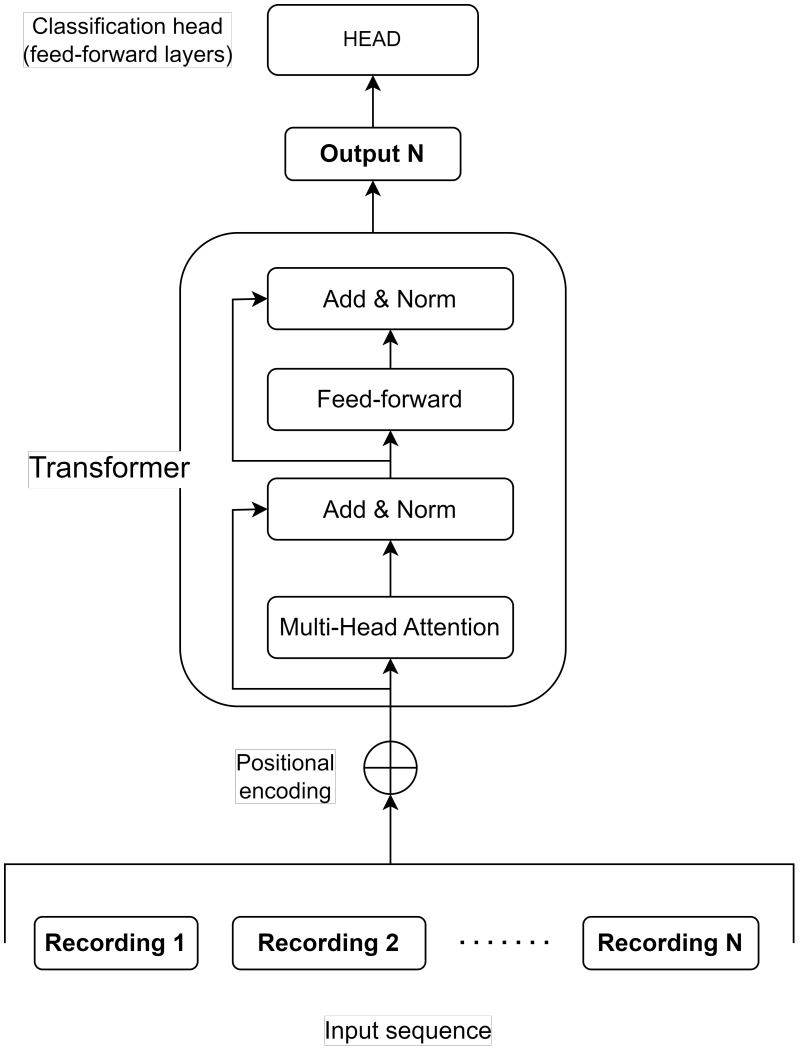

Figure 4.7: The transformer model used for sequential classification of processed disturbance recordings. Consists of positional encoding, transformer encoder and a linear classification layer.

4.3.4 CNN Transformer/LSTM

A more complex neural network was constructed and combined with the sequential model architectures, which focused on analysing the raw current and voltage waveforms. The architecture was a signal processing pipeline combined with a convolutional neural network that processed each recording during the training process in order to learn representative feature vectors. The convolutional head could thus learn to produce embeddings for each waveform appropriate for time series prediction. The signal embedding pipeline connected to the transformer model is illus-

trated in figure 4.8. The figure shows how a sequence of disturbance recordings are processed before fed to the transformer.

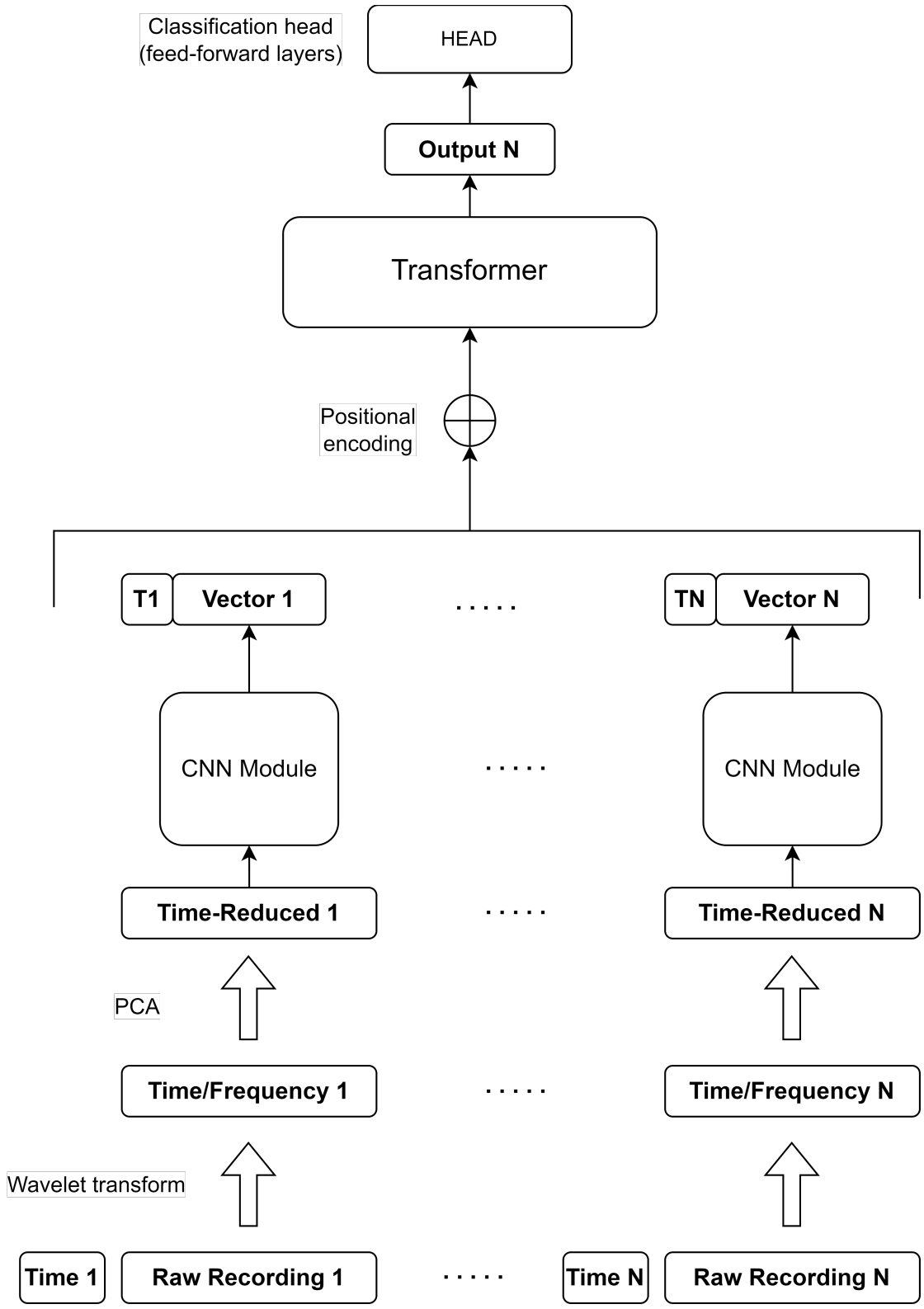

Figure 4.8: The signal processing pipeline connected to the transformer model.

The difficulty with analysis of the raw disturbance recordings, is that the waveforms can have a varying number of measured points, which results in a problem with two variable lengths in the input; The number of disturbance recordings in a sequence, controlled by the sequential model, and the variable length of the recording itself. As explained in previous sections, the sequential model architectures effectively handles the variational length in sequences. The CNN, on the other hand, was implemented utilizing an adaptive pooling layer which transforms the CNN output to a fixed length independent of the input size. However, the batched input signals still needed to have the same input size, and thus padding was required. Padding can drastically affect the performance of the model, which is why the waveforms were processed before fed into the CNN, see figure 4.8. The signal processing layer consisted of a continuous wavelet transform together with principal component analysis for dimensionality reduction. The wavelet transform operation transforms each waveform into an approximate time-frequency domain which has the same number of time segments as the original wave, with an additional frequency domain (see example in figure 4.9). The number of time segments were therefore reduced with PCA before analysed by the CNN. For waves with less data points than the number of components in PCA, the spectrogram were padded by backward repetition across the time-domain. This process minimized the total number of padding values appended to each recording and thus limited the introduced noise. The architecture of the CNN is shown in figure 4.10. As seen in the figure, the CNN feature extractor extracts N features from the reduced spectrograms, which are then passed through the sequential model. In order to make the model time-dependent, the relative time for each recording were appended to each recording after the embedding process.

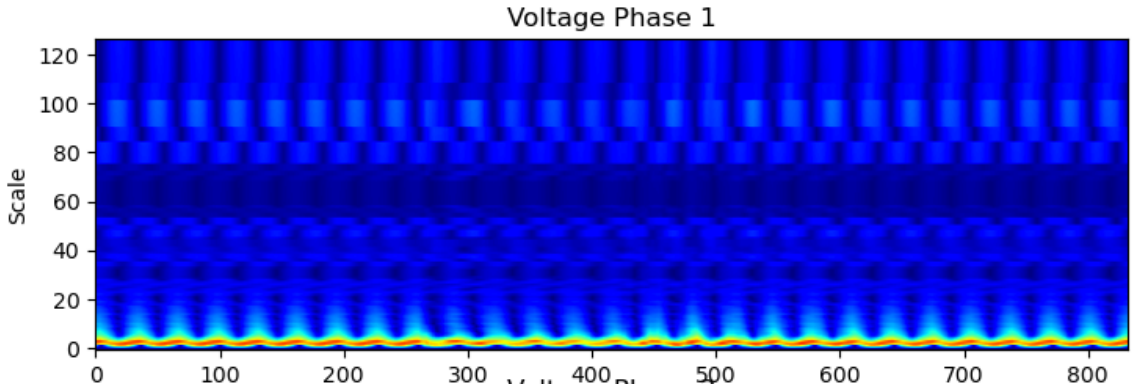

Figure 4.9: Voltage phase 1 spectrogram example. The y-axis is the frequencies, x-axis is the time dimension, and the coloring represents the amplitudes.

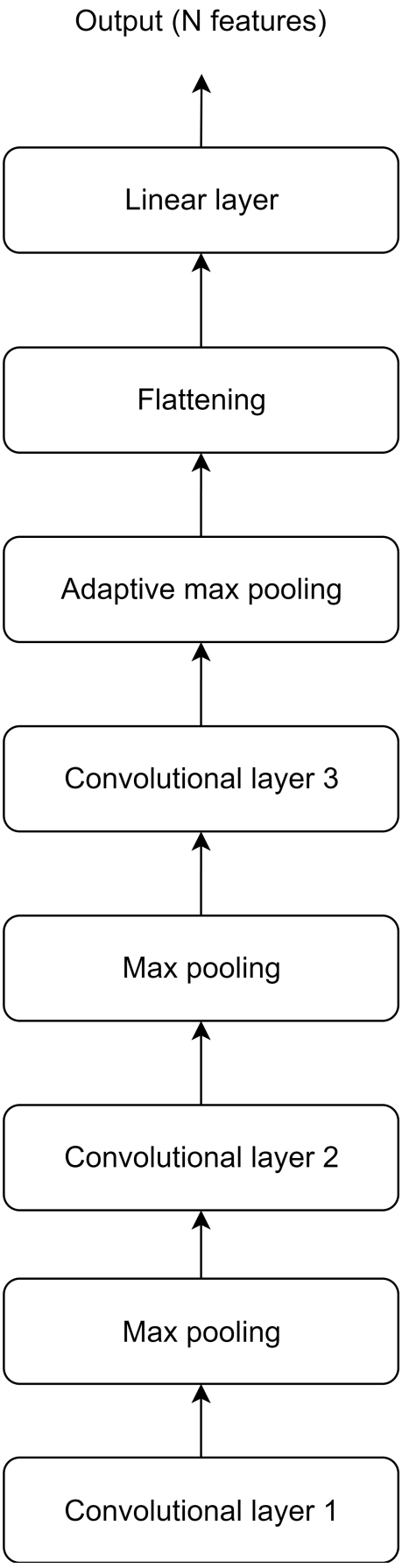

Figure 4.10: The CNN Module in the CNN Transformer architecture.

4.4 Evaluation

For evaluation of the models' performance, the data was split into training and evaluation datasets. However, due to the nature of the problem and the data, it's not enough to randomly split the sequences. By randomly splitting the data sequences, we introduce data leakage between the validation and training sets since the sequences overlap due to how they are processed. This means that the data had to be split by taking time into consideration. Therefore, the evaluation dataset were produced by taking the earliest sequences such that the number of sequences with a positive class were approximately 30 percent of the full dataset, and the rest of the data was selected as training. A gap of one week was left between each of the sets in order to avoid leakage.

5

Results

Results were produced using a hyperparameter grid search for each of the model instances. Learning rate and weight decay were the main hyperparameters optimized with grid search, but some of the models also included an additional parameter whenever it was seen difficult to intuitively with a few experiments decide the value of the parameter. All of the results were provided with the intent of maximizing the performance of each model and thus making good comparisons in order to evaluate the Transformer architecture on a limited time-series classification problem. This includes dataset analysis to optimize each model with respect to the statistical distribution, and nature of the dataset.

5.1 Dataset analysis

The dataset consisted of a total number of 497 disturbance recordings and 23 interruptions across all seven locations. Processing the data into sequences for each location and concatenating resulted in 977 sequences, where 54 sequences had a positive label (an interruption followed the sequence). This gave a positive quota of approximately 0.055. The longest sequence of disturbance recordings had a length of 16. Figure 5.1 illustrates a plot of the disturbance recordings together with interruptions from one of the locations where data have been collected.

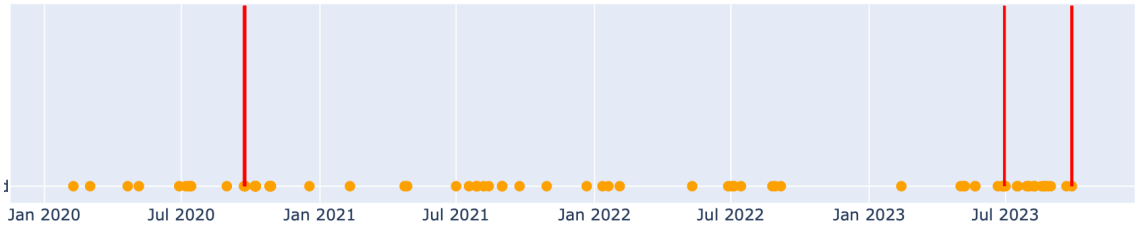

Figure 5.1: Disturbance recordings and interruptions from one of the data locations. The red lines represent the interruptions and the scatters are the disturbance recordings.

Splitting the dataset into training and validation datasets as well as excluding a week of data in between them resulted in the datasets shown in table 5.1 below. Plots

were also produced for the distribution of time until interruption from the point of prediction for each of the datasets (see figure 5.2). The distribution of sequence length for each set is presented in figure 5.3.

Table 5.1: The number of positive and negative samples after train/test split.

	Negative sequences	Positive sequences
Train	571	39
Validation	352	15

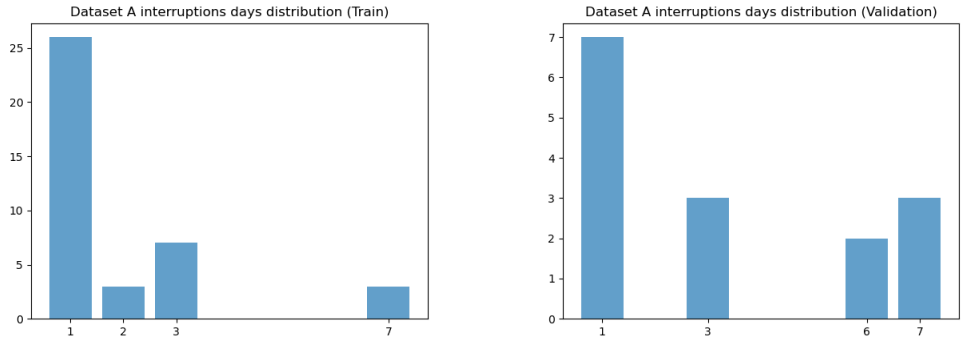

Figure 5.2: Samples (y-axis) with number of days until the interruption (x-axis) from the point of prediction for the positively labeled sequences. The left figure shows the distribution of the training set, while the right figure shows the distribution of the validation data.

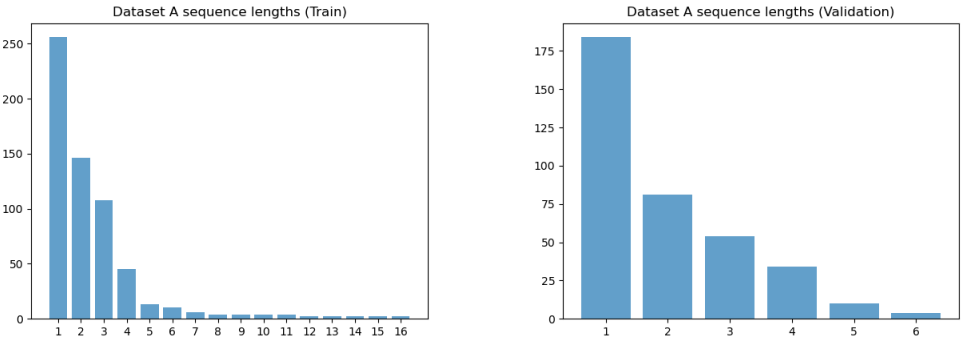

Figure 5.3: The distribution of sequence length for the training and validation sets. The left figure shows the training set while the right figure shows the validation set. Number of data points (y-axis) with different sequence lengths (x-axis).

5.2 Model evaluation

5.2.1 Baseline

The primary baseline tries to classify sequences of disturbance recordings by finding an optimal boundary for the length of the sequence. Minimizing the binary cross entropy with regularization dependent on the boundary, gives a boundary of 3. Therefore, sequences that consists of more than 3 disturbance recordings are classified as positive, else negative. Table 5.2 shows how the boundary and results changes depending on the regularization parameter λ.

Table 5.2: In this table, λ represents the regularization parameter. The table shows the validation scores from using different baseline boundaries controlling how sequences are predicted. Every sequence longer than the boundary are classified as positive.

λ	Boundary	Loss	Ones predicted	Accuracy	F Score	Recall	Specificity
2	3	5.794	0.131	0.839	0.063	0.133	0.869
1.5	4	2.848	0.038	0.921	0.0	0.0	0.96
0.1	15	1.473	0.0	0.959	0.0	0.0	1.0

5.2.2 Functional machine learning model

A functional baseline model was trained on one disturbance recording at a time, each consisting of 370 manually extracted features. It was a feed-forward neural network with two hidden linear layers, consisting of 256 and 128 neurons respectively. Table 5.3 shows the validation scores from training with different learning parameters using grid search.

Table 5.3: Model parameters and validation scores for grid search with a non-sequential feed-forward neural network and the manually extracted features. The highlighted row illustrates the best performing settings, and the bold values are the best scores.

Learning Rate	Weight Decay	Loss	F Score	Recall	Specificity
0.0001	0.001	1.3006	**0.1951**	1.0	0.0149
0.0001	0.01	**1.2960**	0.1928	1.0	0.0
0.0001	0.05	1.3049	0.1928	1.0	0.0
0.00005	0.001	1.2997	0.1928	1.0	0.0
0.00005	0.01	1.3041	0.1928	1.0	0.0
0.00005	0.05	1.3110	0.1928	1.0	0.0
0.000005	0.001	1.3386	0.1928	1.0	0.0
0.000005	0.01	1.3525	0.1928	1.0	0.0
0.000005	0.05	1.3758	0.1928	1.0	0.0

5.2.3 LSTM

The sequential LSTM model were trained using grid search for the learning rate, weight decay, and layers parameters. It was trained on the features manually extracted from the disturbance recordings. Table 5.4 provides the outcome of each training instance of the grid search. The hidden size and the long-term memory had a dimnesion of 64.

Table 5.4: Model parameters and validation scores for grid search on the LSTM model with manually extracted features. The highlighted row illustrates the best performing settings, and the bold values are the best scores.

n_layers	Learning Rate	Weight Decay	Loss	F Score	Recall	Specificity
1	0.0001	0.001	**0.7902**	0.0556	0.0667	0.9432
3	0.0001	0.001	0.8251	**0.1212**	0.1333	0.9545
1	0.0001	0.01	0.8054	0.0870	0.0667	0.9801
3	0.0001	0.01	0.8755	0.0000	0.0000	1.0000
1	0.0001	0.05	0.7970	0.0952	0.0667	0.9858
3	0.0001	0.05	0.8952	0.0000	0.0000	1.0000
1	0.00005	0.001	0.7897	0.0526	0.0667	0.9375
3	0.00005	0.001	0.8264	0.1212	0.1333	0.9545
1	0.00005	0.01	0.8066	0.0870	0.0667	0.9801
3	0.00005	0.01	0.8828	0.0000	0.0000	1.0000
1	0.00005	0.05	0.9059	0.0000	0.0000	1.0000
3	0.00005	0.05	0.9062	0.0000	0.0000	1.0000
1	0.000005	0.001	0.9540	0.0795	0.8667	0.1506
3	0.000005	0.001	0.9206	0.0000	0.0000	1.0000
1	0.000005	0.01	0.9585	0.0695	0.8667	0.0170
3	0.000005	0.01	0.9238	0.0000	0.0000	1.0000
1	0.000005	0.05	0.9673	0.0794	1.0000	0.0114
3	0.000005	0.05	0.9286	0.0000	0.0000	1.0000

Prediction plots that shows how the probability of a future interruption change over time were produced for each data location for the best performing model. Figure 5.4 illustrates the moving probability over time from one of the data location, for both the training and validation data. Since the data is split into training and validation sets such that the earliest sequences are used for evaluation, the plots illustrates the moving probability over the validation set in the left figure and the probability over the training set in the right figure. For the other data locations, see appendix. The confusion matrices for predictions are presented in figure 5.5.

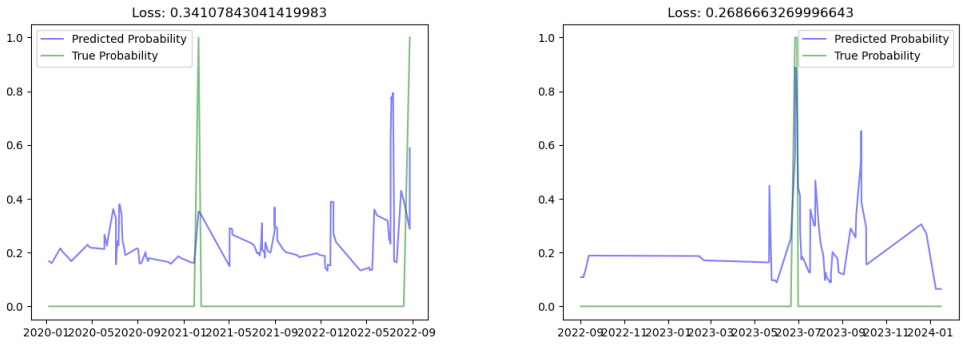

Figure 5.4: The LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

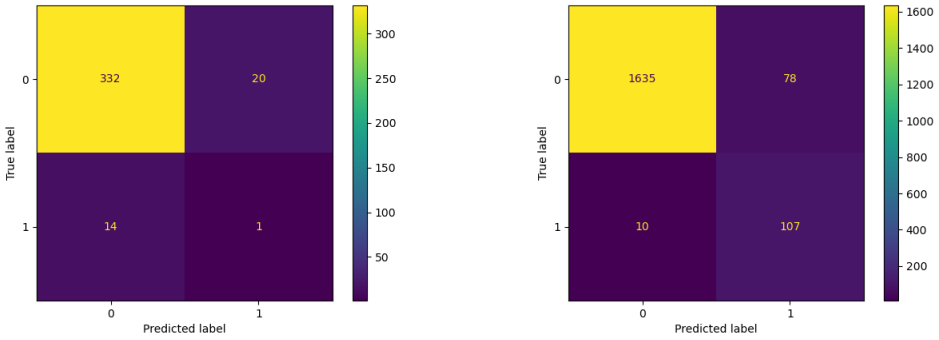

Figure 5.5: The confusion matrices for predictions on the validation and training data using the LSTM model architecture. The left figure is predictions on the validation set, while the right figure represents the training set. Here, 1 refers to a positive sequence while 0 refers to a negative sequence.

5.2.4 Transformer

The results obtained when training and evaluating the transformer using the manually extracted features are illustrated in table 5.5. The results were obtained by grid search on the batch-size, learning rate, and weight decay parameters. The model was trained over 500 epochs and the model consisted of one transformer layer with one attention head. The feed-forward dimension was 64 and the dropout probability was 0.01. The prediction plots and confusion matrices are presented in figures 5.6 and 5.7 respectively.

Table 5.5: Model parameters and validation scores for grid search on the transformer model with manually extracted features. The highlighted row illustrates the best performing settings, and the bold values are the best scores.

Batch Size	Learning Rate	Weight Decay	Loss	F Score	Recall	Specificity
1830	0.0001	0.001	0.8280	0.1935	0.2	0.9631
1830	0.0001	0.01	0.7701	0.25	0.3333	0.9432
1830	0.0001	0.05	0.7398	0.1667	0.1333	0.9801
1830	5e-05	0.001	0.8371	0.2143	0.2	0.9716
1830	5e-05	0.01	0.7871	0.2439	0.3333	0.9403
1830	5e-05	0.05	0.7414	0.2222	0.2	0.9744
1830	5e-06	0.001	0.8543	0.2069	0.2	0.9688
1830	5e-06	0.01	0.8507	0.1818	0.2	0.9574
1830	5e-06	0.05	0.8994	0.1224	0.2	0.9119
457	0.0001	0.001	**0.7105**	0.2174	0.3333	0.9261
457	0.0001	0.01	0.7224	0.25	0.2	0.9830
457	0.0001	0.05	0.7428	**0.2963**	0.2667	0.9773
457	5e-05	0.001	0.7144	0.2759	0.2667	0.9716
457	5e-05	0.01	0.7203	0.2581	0.2667	0.9659
457	5e-05	0.05	0.7433	0.25	0.2	0.9830
457	5e-06	0.001	0.7747	0.2	0.1333	0.9915
457	5e-06	0.01	0.7945	0.2222	0.1333	0.9972
457	5e-06	0.05	0.8292	0.2353	0.1333	1.0

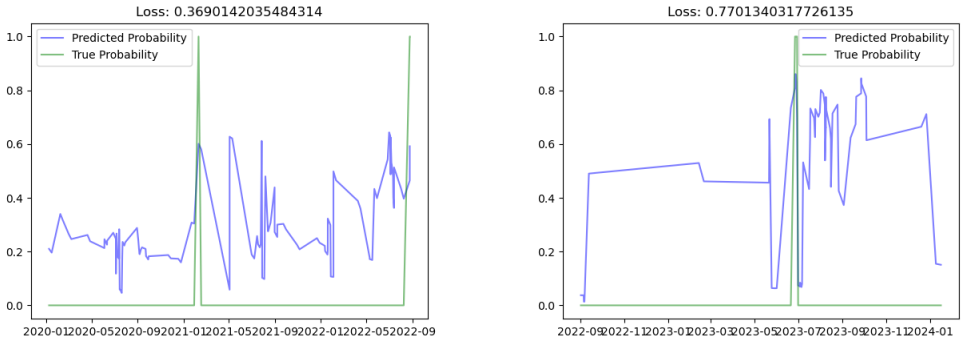

Figure 5.6: The transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

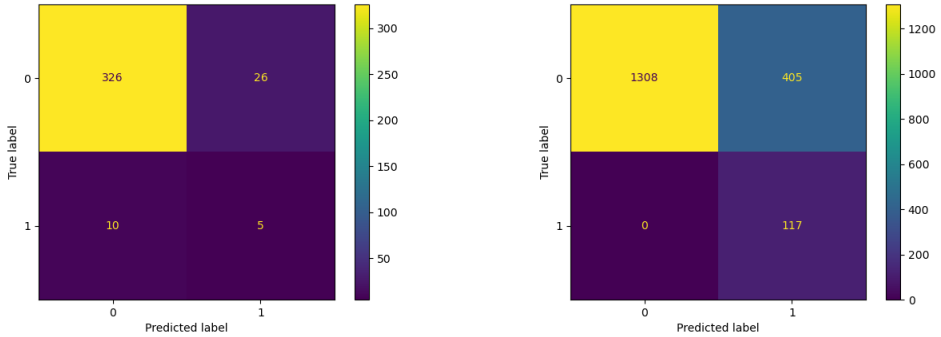

Figure 5.7: The confusion matrices for predictions on the validation and training data using the transformer model architecture. The left figure is predictions on the validation set, while the right figure represents the training set. Here, 1 refers to a positive sequence while 0 refers to a negative sequence.

5.2.5 Transformer and LSTM with pre-training

To further improve the results of the LSTM and Transformer models using the manually extracted features, they were both evaluated by first pre-training on the similar dataset B, and then fine-tuned on the target dataset A. Both of the models used the same settings as previously described in section 5.2.3 and 5.2.4. The results for the transformer are provided in table 5.6 with prediction plots and confusion matrices in 5.8 and 5.9. Similar for the LSTM, the results, and plots are shown in 5.7, 5.10, and 5.11.

Table 5.6: Model parameters and validation scores for grid search on the pre-trained transformer model using the manually extracted features. The highlighted row illustrates the best performing settings, and the bold values are the best scores.

Batch size	Learning Rate	Weight Decay	Loss	F Score	Recall	Specificity
1830	0.0001	0.001	**0.66844**	**0.31429**	0.73333	0.875
1830	0.0001	0.01	0.73162	0.15686	0.26667	0.90909
1830	0.0001	0.05	0.74297	0.14035	0.26667	0.89205
1830	5e-05	0.001	0.74990	0.21053	0.13333	0.99432
1830	5e-05	0.01	0.71463	0.14545	0.26667	0.89773
1830	5e-05	0.05	0.73144	0.10526	0.20000	0.88920
1830	5e-06	0.001	0.83734	0.15000	0.20000	0.93750
1830	5e-06	0.01	0.85399	0.10256	0.13333	0.93750
1830	5e-06	0.05	0.88902	0.15873	0.33333	0.87784
457	0.0001	0.001	0.78795	0.17391	0.13333	0.98295
457	0.0001	0.01	0.79487	0.18182	0.13333	0.98580
457	0.0001	0.05	0.74882	0.20408	0.33333	0.91761
457	5e-05	0.001	0.74533	0.17391	0.26667	0.92330
457	5e-05	0.01	0.75254	0.20513	0.26667	0.94318
457	5e-05	0.05	0.74125	0.22857	0.26667	0.95455
457	5e-06	0.001	0.80075	0.14815	0.13333	0.97159
457	5e-06	0.01	0.81086	0.13793	0.13333	0.96591
457	5e-06	0.05	0.83624	0.11111	0.13333	0.94602

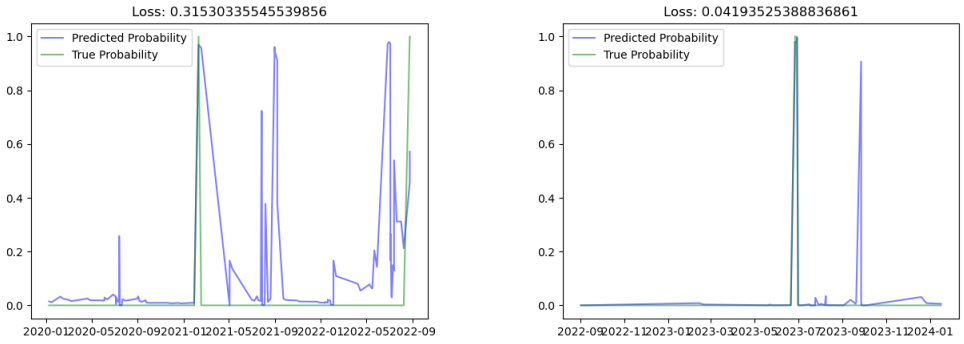

Figure 5.8: The pre-trained transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

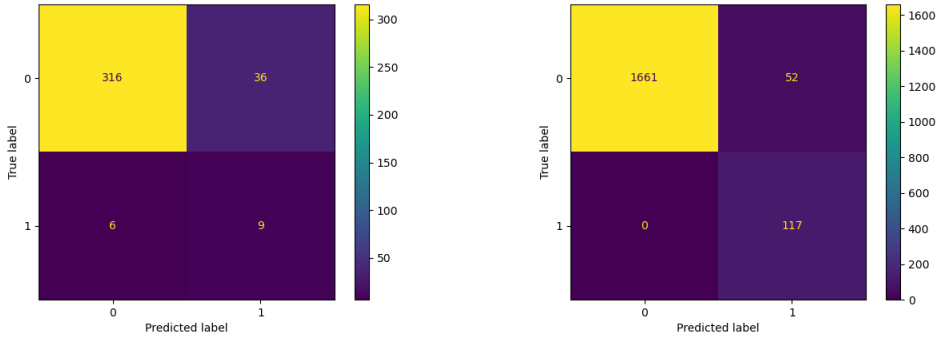

Figure 5.9: The confusion matrices for predictions on the validation and training data using the pre-trained transformer model. The left figure is predictions on the validation set, while the right figure represents the training set. Here, 1 refers to a positive sequence while 0 refers to a negative sequence.

Table 5.7: Model parameters and validation scores for grid search on the pre-trained LSTM model using the manually extracted features. The highlighted row illustrates the best performing settings, and the bold values are the best scores.

Learning Rate	Weight Decay	Loss	F Score	Recall	Specificity
0.0001	0.001	**0.7114**	**0.2381**	0.3333	0.9375
0.0001	0.01	0.7878	0.1143	0.1333	0.9489
0.0001	0.05	0.8383	0.0769	0.0667	0.9716
0.00005	0.001	0.7200	0.1951	0.2667	0.9375
0.00005	0.01	0.8046	0.1143	0.1333	0.9489
0.00005	0.05	0.9202	0.1260	0.5333	0.7045
0.000005	0.001	1.2293	0.0785	1.0000	0.0000
0.000005	0.01	1.2370	0.0785	1.0000	0.0000
0.000005	0.05	1.2700	0.0785	1.0000	0.0000

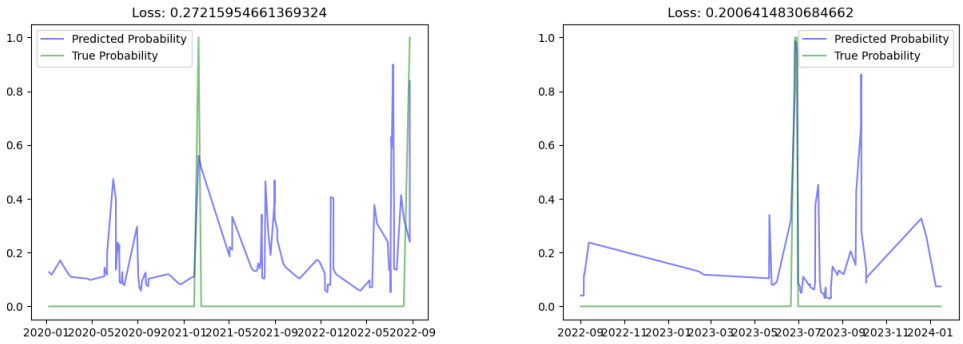

Figure 5.10: The pre-trained LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

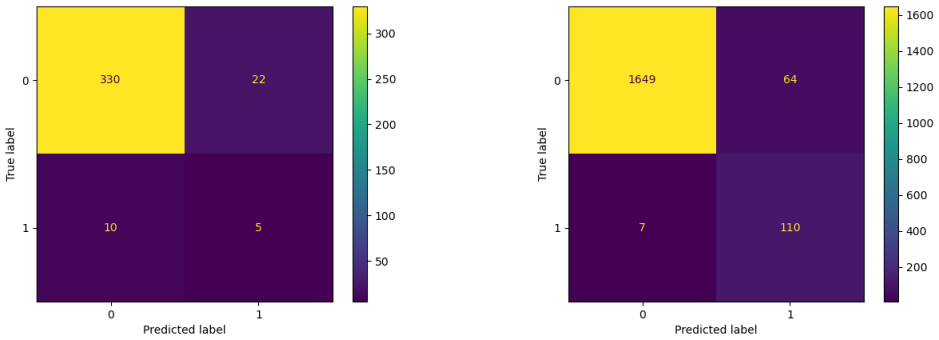

Figure 5.11: The confusion matrices for predictions on the validation and training data using the pre-trained LSTM model. The left figure is predictions on the validation set, while the right figure represents the training set. Here, 1 refers to a positive sequence while 0 refers to a negative sequence.

5.2.6 CNN LSTM

The CNN architecture which analyzed PCA reduced spectrograms of the waveforms used three convolutional layers with 3×3 kernel sizes and $(1, 1)$ stride, with pooling kernels size and stride of 2×2 and $(2, 2)$. Each of the convolutional layers had 6, 12 and 24 kernels respectively. The dimension of latent space produced by the CNN was set to 199, and appending the time feature resulted in a feature vector with a total dimension of 200. The LSTM model connected to the CNN architecture had the same structure as the LSTM which did inference using the manually extracted feature vectors, described in section 5.2.3. However, this LSTM module used one layer, which was shown to perform the best in section 5.2.3. Table 5.8 shows the results obtained by grid search for learning rate and weight decay on the CNN LSTM model. The prediction plots and confusion matrices for the validation and training sets are provided in figures 5.12 and 5.13.

Table 5.8: Model parameters and validation scores for grid search on the CNN LSTM model. The highlighted row illustrates the best performing settings, and the bold values are the best scores.

Learning Rate	Weight Decay	Loss	F Score	Recall	Specificity
0.0001	0.001	**0.6601**	**0.1707**	0.4667	0.8305
0.0001	0.01	0.6846	0.1235	0.3333	0.8277
0.0001	0.05	0.6873	0.1149	0.3333	0.8107
0.00005	0.001	0.6733	0.1316	0.3333	0.8418
0.00005	0.01	0.6901	0.1149	0.3333	0.8107
0.00005	0.05	0.6914	0.1176	0.3333	0.8164
0.000005	0.001	0.6853	0.1149	0.3333	0.8107
0.000005	0.01	0.6889	0.1176	0.3333	0.8164
0.000005	0.05	0.7874	0.1169	0.6000	0.6328

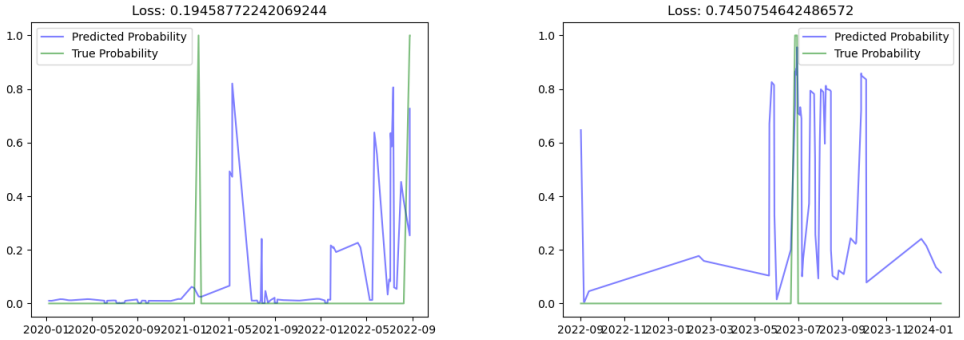

Figure 5.12: The CNN LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

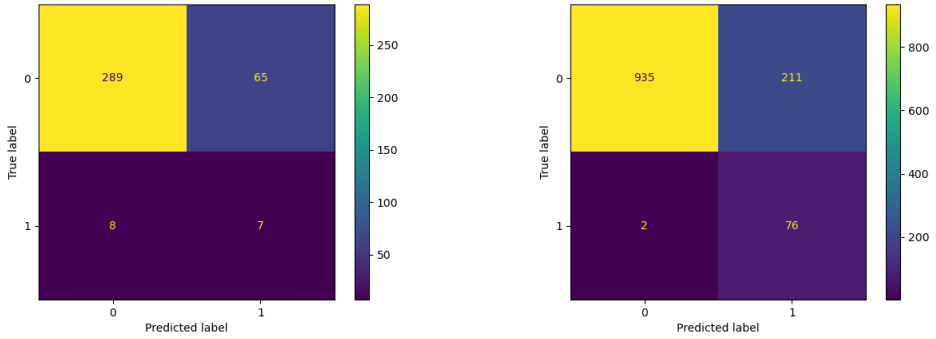

Figure 5.13: The confusion matrices for predictions on the validation and training data using the CNN LSTM model architecture. The left figure is predictions on the validation set, while the right figure represents the training set. Here, 1 refers to a positive sequence while 0 refers to a negative sequence.

5.2.7 CNN Transformer

For training and prediction of disturbance recordings as raw waveforms using the CNN transformer architecture, the results obtained are shown in table 5.9. The CNN part of this model structure used the same architecture as for the LSTM, described in section 5.2.6. The transformer architecture was the same as for inference with the transformer on the manually extracted features in section 5.2.4. The prediction plots and confusion matrices follows in figures 5.14 and 5.15.

Table 5.9: Model parameters and validation scores for grid search on the CNN transformer model. The highlighted row illustrates the best performing settings, and the bold values are the best scores.

Learning Rate	Weight Decay	Loss	F Score	Recall	Specificity
0.0001	0.001	0.5863	0.1224	0.6000	0.6525
0.0001	0.01	0.5767	0.2264	0.4000	0.9096
0.0001	0.05	0.6746	0.1208	0.6000	0.6469
0.00005	0.001	0.6003	0.1154	0.6000	0.6271
0.00005	0.01	**0.5738**	**0.3043**	0.4667	0.9322
0.00005	0.05	0.6742	0.1200	0.6000	0.6441
0.000005	0.001	0.6800	0.1154	0.6000	0.6271
0.000005	0.01	0.6853	0.1154	0.6000	0.6271
0.000005	0.05	0.6932	0.1154	0.6000	0.6271

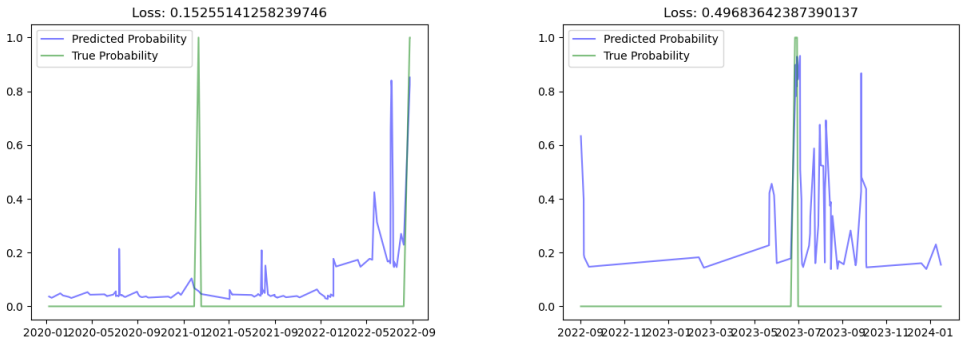

Figure 5.14: The CNN transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

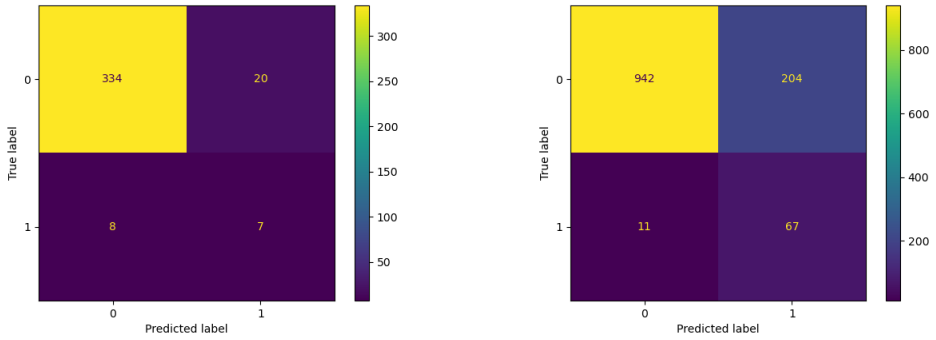

Figure 5.15: The confusion matrices for predictions on the validation and training data using the CNN transformer model. The left figure is predictions on the validation set, while the right figure represents the training set. Here, 1 refers to a positive sequence while 0 refers to a negative sequence.

6

Conclusion

6.1 Discussion

This discussion section includes discussion regarding the results presented in section 5, as well as reasoning for the model architectures.

6.1.1 Data analysis

By looking at the distribution and scale of the target dataset shown in section 5.1, it is clear that the dataset is very small and limited. With only approximately 5 percent of the sequences having a positive label, the problem becomes extremely unbalanced and difficult, which shows the importance of how the data is processed, such as data augmentation and sequence processing. As seen in table 5.1, the validation dataset only includes 15 positive sequences out of 367, which means that the positive class constitutes to only 4 percent of all sequences. This will thus have a large impact on how evaluation scores are interpreted. For instance, accuracy is in this case not a good metric since predicting all sequences as negative would give a high accuracy score even though the model is not performing well. Regarding F score, successfully predicting 3 out of 15 positive sequences with very few false positives does not yield a large F Score, but can still be seen as a good result with respect to the class imbalance and the difficult nature of the problem. The large class imbalance also contributes to the difficulty of splitting the dataset into validation and training sets but still containing the statistical distribution in both sets. The data split were also required to happen with respect to time in order to avoid data leakage between the two sets, because the sequences close to each other in time naturally overlaps. Thus, splitting the dataset with respect to time and number of positive sequences was necessary, and yielded the distribution of days until interruption seen in figure 5.2. It should be noted that this result came from using the earliest sequences as validation and the later as training, and that processing the data the other way around did not yield as equal distributed days-until-interruption between the two sets. This problem also eliminated the possibility of cross-validation, since it was not guaranteed to obtain good enough splits.

Large plots were produced for each manually extracted feature for each location in dataset A. Figure 5.1 is an example of such a plot for one feature and location, where it can be clearly seen that the number of disturbance recordings are increased

as the time is approaching the time for an interruption. This strongly indicates a correlation between the number of disturbance recordings which shows the importance of comparing the results with the baseline model. For a few features and locations, the plots showed a value difference dependent on time, see figure 6.1. The coloring of each disturbance recording illustrates the value of the specific feature, and it can clearly be seen that the value of this feature permanently changes in January 2022. This could be due to manual maintenance causing these changes. This removes the time independence, which can cause biased results when creating training and validation sets based dependent on time. However, this was only noticed on a few features and therefore might not drastically affect the results.

Figure 6.1: Plot of disturbance recordings for one location and feature. The coloring of each disturbance recordings represents the value of the feature at each recording.

In the figure showing the distribution of sequence length in dataset A (5.3), it is clear that most sequences are very short and consists of very few recordings. Most sequences has a length of one to three recordings, which can be seen as a disadvantage when utilizing an attention-based model. That is because attention-based models are specialized in finding patterns between several input tokens.

6.1.2 Model architectures

The transformer architecture is generally good for inference on natural language data, thus, one can expect the model to perform well for sequential data in other areas as well. Words in NLP are usually embedded as feature vectors during the training process so that feature vectors are fed into the transformer. In this study, we are also working with feature vectors, which comes from signals instead of words. However, the data in this study is very limited compared to usual tasks in NLP, which is why the model size had to be limited in order to prevent overfitting and increase generalization. The choice of using one attention head with one transformer encoder therefore comes from the limited amount of data. Transformers often requires much training data to work well, and increasing the number of attention heads drastically increases the number of learnable parameters. Thus, the smaller transformer model with one attention head and one layer was found to be most effective for the dataset used in this study. Another deviation compared to NLP, is that this time-series classification problem includes time, which is why a relative time feature were appended to each of the feature vectors in order to let the model

catch the relevancy of each recording. This means that a time-window with a single recording can affect the predicted probability depending on where it's located in the window.

The thought behind the more complex model architectures (CNN LSTM, CNN transformer) is the same as for the transformer performing inference on the manually extracted features, but the feature vectors are instead learned during the training process from the raw current and voltage signal data. The waveform data introduced two variable length inputs and the model therefore had to be able to handle variational input sizes in two dimensions, which made it especially important how padding was applied. Applying padding in different ways might have a big impact on the performance of the model. Because of the large deviation in signal length (varying from below 100 data points up to 8000), padding and signal processing needed to be carefully considered. By transforming the waveforms using continuous wavelet transform into a trade-off between time and frequency and then reducing the time dimension from the length of each signal into 64 with principal component analysis resulted in time-frequency images of equal sizes. This made sure that each image kept the most important time segments as well as effectively turned each transformed signal into equal sizes. However, for the disturbance recordings smaller than 64 data points, the PCA step was skipped and padding was applied by repeating the last time-segments multiple times, which resulted in minimal padding, as well as avoidance of zero-padding. This approach of signal processing allows for high customizability in terms of how much the waveforms should be generalized and was therefore thought to perform well even for very limited datasets. For the same reason as for the transformer model used for inference on the manually extracted features, a time feature was added to the latent space provided by the CNN module, in order to let the model be time-dependent.

6.1.3 Results

By first looking at the results provided by the baseline machine learning algorithm, we can see that the algorithm does not make any successful predictions until the boundary regularization parameter is large enough such that the boundary becomes 3, see table 5.2. This is not surprising since there are very few sequences of recordings longer than 4, as illustrated in figure 5.3. Thus, we can understand that the longer sequences have negative labels, which contradicts the theory that there is a direct correlation between the number of disturbance recordings and upcoming faults in electrical power systems. The low precision and F scores indicates that there are very few true positives (most likely one), with several false positives and false negatives. These scores therefore also indicate that there are positive and negative sequences on both sides of the boundary.

Observing the results from the seconds baseline model, the feed-forward neural network, which classifies single disturbance recordings, it performs very poorly (see table 5.3). As can be seen in the table, the model optimizes its loss by simply learning the weighted loss of the positive predictions in the loss function, and thus

predicts every sample as positive. Figure 6.2 clearly illustrates how the evaluation and training losses stops decreasing with a larger number of epochs. This is most likely because the problem is too complex for the feed-forward network and implies that the network has nothing more to learn from the data. The results obtained from this model thus indicate that it's not enough to simply look at single disturbance recordings, but rather sequences of them to find correlating patterns.

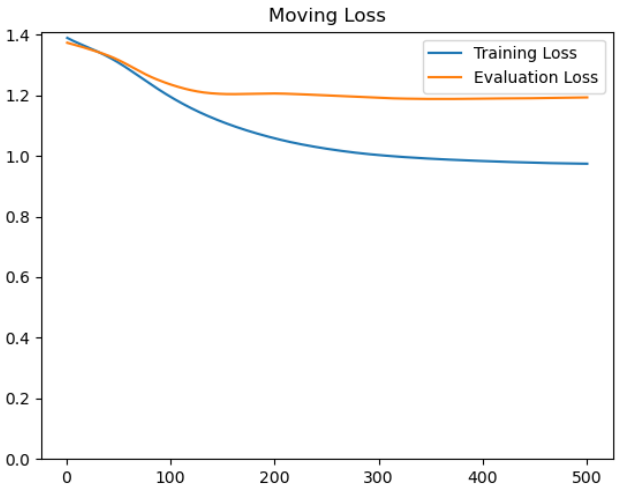

Figure 6.2: Moving loss (y-axis) with respect to epochs (x-axis) for the functional feed-forward neural network.

Both of the baseline models find trivial solutions that does not result in any particular correlation between disturbance recordings and faults, but comparing them to the LSTM and Transformer models show how the more advanced architectures find patterns in the data instead of falling back on a trivial solution. The results from the LSTM and Transformer models trained on the manually extracted features (see table 5.4 and 5.5) both perform better than the baseline solutions, which means that they actually learn from the input vectors. However, looking at the results from the LSTM, we can see that the feed-forward baseline model achieves a higher F score, which is because of the high recall score obtained from predicting every sample as positive. This is why the LSTM model's performance is still considered better; it makes very few predictions and thus avoid false negatives (high specificity score). Regarding the low recall score, it's because the model predicts a lot of positive sequences as negative, which means that it predicts several faults as a non-fault. Although, in the case of electrical power systems, successfully predicting one out of many faults is better than predicting all of them as positive. Comparing the LSTM scores to the scores produced by the transformer, the transformer performs better, which is shown by the F scores and loss values. The specificity scores are very similar for both of the architectures, but the transformers improved recall score suggests that the model more accurately predicts true faults compared to the LSTM. In the context of fault prediction in electrical power systems, another important aspect is the illustration of the prediction probability over time. The predicted probability

is generally expected to be very low until an interruption enters the labeling window. However, comparing both of the models, it's noticed that the LSTM predicts a lower and more consistent probability for negative sequences (see figure 5.4 and 5.6). A theory to why this happens is because the transformer adopts to a higher and more inconsistent probability in order to decrease the risk of predicting a low probability for positive samples, since they are weighted in the loss function by a factor 10. Even though the transformer produce better scores, this phenomenon is not desired because of the reduced interpretability of the model's output. The probabilistic output should be a measure of the model's confidence in the prediction, which is not the case for the transformer. Figure 6.4 illustrates another example of this behaviour on training data, where the LSTM (left figure) and the transformer (right figure) is compared. The figures show similar patterns for both of the models, although the transformer output more fluctuating probabilities, which most likely is due to the fact the the transformer is more complex with more learnable parameters compared to the LSTM.

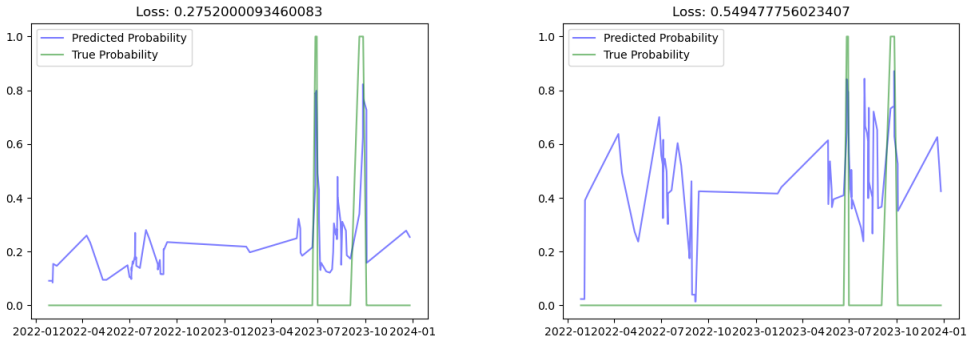

Figure 6.3: Prediction plots for the LSTM (left) and transformer (right) on training data. The figure shows the predicted probability (y-axis) over time (x-axis). NOTE: The loss presented in the figure is not weighted.

Comparing the LSTM and transformer results obtained by first pre-training on dataset B (table 5.7 and 5.6), we can see a drastic improvement for both of the models compared to not utilizing pre-training. Looking at the confusion matrices (see figure 5.11 and 5.9), we see that the pre-trained transformer manages to successfully predict 9 out of 15 faults in the validation set, compared to before, where the transformer predicted only 5 faults. This improvement is also applied to the LSTM architecture, where the successful predictions changes from 1 to 5. These improvements comes with an increased number of false positive predictions, but the models' scores still overall improve. The transformer architecture reaches an F score of approximately 0.31 while the LSTM gets an F score of 0.24. Thus, we can still see that the transformer performs better than the LSTM in terms of scores. By looking at the prediction graphs in the following figure, and comparing to the previous figures, it's clear that the transformer has now learned to make more confident predictions, while the predicted probabilities for the LSTM mostly remains the same.

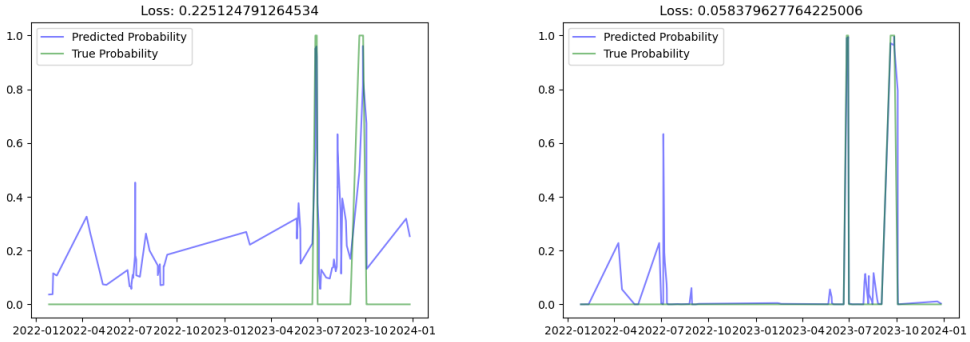

Figure 6.4: Prediction plots for the pre-trained LSTM (left) and pre-trained transformer (right) on training data. The figure shows the predicted probability (y-axis) over time (x-axis). NOTE: The loss presented in the figure is not weighted.

Reasoning around this, it's likely that it happens because the transformer learns something from dataset B that it cannot learn entirely from dataset A. The datasets are very similar in terms of data structure, however, one thing that sets them apart is the interval of disturbance recordings. Dataset B contains more disturbance recordings appearing in closer time intervals, which naturally yields longer sequences during processing of the data. Figure 6.5 illustrates the training sequence length distribution for each of the datasets. The transformer architecture is specifically efficient in finding correlating patterns between input tokens because of the attention mechanism, and pre-training on a dataset that contains much longer input sequences allows for the model to more efficiently learn patterns between input recordings and thus improve its performance. This study used an input window of one week, but looking at the results of this, it could have been beneficial to extend the input window and thus allow the model to train on longer input sequences.

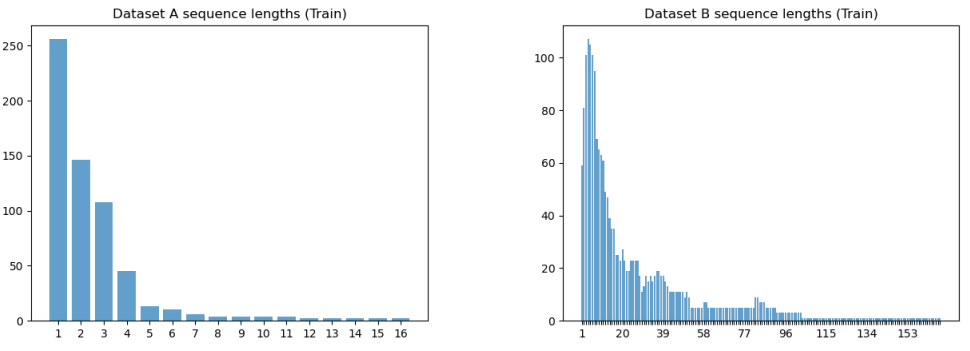

Figure 6.5: The training distribution of sequence length for dataset A and B. The left figure shows the sequence length for dataset A, while the right figure shows the distribution for dataset B. The figures represents the number of data points (y-axis) with each sequence length (x-axis).

It is concluded that the short sequences contained in dataset A limits the results

of the transformer, but another question is whether it's also limited by the manually extracted features, and if it would perform better in a more general setting of time series prediction. The results from the CNN LSTM and CNN transformer architectures (see table 5.8 and 5.9) show clear improvement over the architectures without the CNN head. The CNN transformer got a best F score of approximately 0.30 with a much lower loss, which even compares to the pre-trained transformer model. This illustrates that there is a bottleneck in the manually extracted features and by comparing the recall and F score of the pre-trained transformer with the CNN transformer, the precision is higher on the CNN transformer. This means that the CNN transformer has a reduced number of false positive predictions and an increased number of false negative predictions compared to the pre-trained model (see figure 5.15). We can also see in the prediction graphs (figure 5.14) that the CNN transformer's predictions are more confident and the output is therefore more interpretable than the model using the manually features. But comparing the prediction graph with the graph of the pre-trained transformer, the limited sequence lengths still seem to penalize the performance.

6.2 Summary

Evaluating the overall performance of the attention based transformer model for time series prediction in electrical power systems with significant limitations on data, it clearly performs better than a non-attention based recurrent neural network. From the produced results, we can notice that the transformer's performance is highly affected by the training data and how it's processed. The data does not seem to have as big impact on the LSTM as on the transformer, which shows that the processing of the data is very important when utilizing an attention based model. Therefore, in time series prediction problems where data is very limited, it's crucial to formulate the problem and process the data such that the number of data points are maximized, as well as obtaining appropriate sequence lengths. This is necessary in order to fully utilize the attention mechanism and the power of the transformer architecture.

6.3 Risk Analysis and Ethical Considerations

Regarding risks of this project, it's difficult to come up with reasons as to why it would introduce a risk to potentially predict future faults in electrical power systems. In worst case scenario, the prediction system incorrectly predicts the probability of a fault, which simply almost equates to not having a system at all. Therefore, if the system comes with many false-negative misclassifications, it would not affect the user in a costly or negative way. On the other hand, if the system tend to misclassify data as false-positive, it could potentially lead to managers sending operators for maintenance when it's not necessary. Regarding the data used for inference, it's not in any way sensitive and thus does not exploit any private or personal information that can do any harm.

Another aspect to consider is the possibility of using the system to exploit the power grid. The system can be used to find potential faults in the grid, but information about where and when something in a power system can go wrong is not considered sensitive information and therefore cannot be exploited by malicious actors.

The only ethical point relevant to consider with this project are the jobs that the system could replace. However, this is very unlikely since operators will still be required in order to fix the potential electrical interruption. This system does not prevent or resolve interruptions, but only provide insight in when an interruption happen, so that it can be prevented beforehand.

6.4 Further work

In this study, we have seen that an attention-based transformer performs significantly better than an LSTM in the context of time series prediction in electrical power systems. However, there are several techniques and methods that can be utilized in future work to possibly further improve the inference on limited and unbalanced data with an attention-based transformer. This includes investigating ways to modify the problem formulation or approach in such a way that inference can be made over a larger amount of data. For instance, waveform emebeddings can be trained separate from the classification model by using an denoising autoencoder. This can be done by adding noise to the waveforms in different ways, and then letting an autoencoder restore the original signals. This approach could potentially yield improved signal embeddings as data can effectively be increased by coming up with different ways to introduce noise. The signal embedding module can then be attached to the classification model for inference. The pre-trained signal embedding module may also be fine-tuned during training of the classification model, or the weights can be frozen in order to prevent modified embeddings. Another approach to further improve the results on a problem of this context would be to directly introduce denoising techniques in the classification model, and use it for pre-training. Given manually extracted features, or features from a pre-trained waveform embedder, the classification model can be pre-trained by introducing noise directly into the sequence of feature vectors, and then letting the model restore the original sequence. Example of noise could be swapping or masking disturbance recordings in a sequence. This approach could potentially help in pre-training the classification model without having to find an additional and similar dataset.

Apart from the mentioned improvements that could be investigated, it would also be interesting to see how the transformer architecture performs on other types of time series prediction tasks with similar context, compared to the LSTM architecture. It would also be interesting to further investigate when an LSTM model architecture would be preferred over the transformer in terms of time series prediction.

Bibliography

[1] H. Ritchie, P. Rosado, and M. Roser. "Energy production and consumption." (), [Online]. Available: https://ourworldindata.org/energy-production-consumption.

[2] Iea. "Electric vehicles data." (), [Online]. Available: https://www.iea.org/energy-system/transport/electric-vehicles.

[3] R. Anil, S. Borgeaud, and Y. Wu, "Gemini: A family of highly capable multimodal models," Google, Tech. Rep., 2023. [Online]. Available: https://arxiv.org/abs/2312.11805.

[4] E. BALOUJI, K. BÄCKSTRÖM, V. OLSSON, *et al.*, "Distribution network fault prediction utilising protection relay disturbance recordings and machine learning," Eneryield. ABB. Vaasan Sähköverkko, Tech. Rep., 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10267272.

[5] K. W. Høiem, "Predicting fault events in the norwegian electrical power system using deep learning - a sequential approach," Norwegian University of Life Science, Tech. Rep., 2019. [Online]. Available: https://nmbu.brage.unit.no/nmbu-xmlui/handle/11250/2608290.

[6] P. Bilokon and Y. Qiu, "Transformers versus lstms for electronic trading," Tech. Rep., 2023. [Online]. Available: https://arxiv.org/abs/2309.11400.

[7] Wikipedia. "Electric power system." (), [Online]. Available: https://en.wikipedia.org/wiki/Electric_power_system.

[8] Wikipedia. "Alternating current." (), [Online]. Available: https://en.wikipedia.org/wiki/Alternating_current.

[9] G. Woo, C. Liu, A. Kumar, C. Xiong, S. Savarese, and D. Sahoo, "Unified training of universal time series forecasting transformers," Tech. Rep., 2024. [Online]. Available: https://arxiv.org/abs/2402.02592.

[10] S. Brown. "Machine learning, explained." (2021), [Online]. Available: https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained.

[11] B. Mehlig., *Machine Learning with Neural Networks, An Introduction for Scientists and Engineers*. Cambridge University Press, 2021.

[12] Wikipedia. "Artificial neural networks." (), [Online]. Available: https://en.wikipedia.org/wiki/Artificial_neural_network.

[13] P. Diederik and J. B. Kingma, "Adam: A method for stochastic optimization," Tech. Rep., 2014. [Online]. Available: https://arxiv.org/abs/1412.6980.

[14] A. F. Agarap, "Deep learning using rectified linear units (relu)," Tech. Rep., 2018. [Online]. Available: https://arxiv.org/abs/1803.08375.

[15] J. Schmidhuber, "Deep learning in neural networks: An overview," Tech. Rep., 2014. [Online]. Available: https://arxiv.org/pdf/1404.7828.

[16] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," Tech. Rep., 2019. [Online]. Available: https://arxiv.org/abs/1912.05911.

[17] G. Giacaglia. "How transformers work." (), [Online]. Available: https://towardsdatascience.com/transformers-141e32e69591.

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory," Tech. Rep., 1997. [Online]. Available: https://www.bioinf.jku.at/publications/older/2604.pdf.

[19] M. Varikuti. "Lstm networks." (), [Online]. Available: https://medium.com/mlearning-ai/lstm-networks-75d44ac8280f.

[20] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," Google, Tech. Rep., 2017. [Online]. Available: https://arxiv.org/abs/1706.03762.

[21] Y. Liu. "Intrusion detection model based on improved transformer." (), [Online]. Available: https://www.researchgate.net/figure/Transformer-classification-structure_fig4_370909988.

[22] S.-H. Kim, Z. W. Geem, and G.-T. Han, "Hyperparameter optimization method based on harmony search algorithm to improve performance of 1d cnn human respiration pattern recognition system," Google, Tech. Rep., 2020. [Online]. Available: https://www.researchgate.net/publication/342632756_Hyperparameter_Optimization_Method_Based_on_Harmony_Search_Algorithm_to_Improve_Performance_of_1D_CNN_Human_Respiration_Pattern_Recognition_System.

[23] M. Hossin, "A review on evaluation metrics for data classification evaluations," Tech. Rep., 2015. [Online]. Available: https://www.researchgate.net/publication/275224157_A_Review_on_Evaluation_Metrics_for_Data_Classification_Evaluations.

[24] S. Narkhede. "Understanding confusion matrix." (), [Online]. Available: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62.

[25] O. Rainio, J. Teuho, and R. Klén, "Evaluation metrics and statistical tests for machine learning," Tech. Rep., 2024. [Online]. Available: https://www.nature.com/articles/s41598-024-56706-x.

[26] F. Zhuang, Z. Qi, K. Duan, *et al.*, "A comprehensive survey on transfer learning," Tech. Rep., 2020. [Online]. Available: https://arxiv.org/abs/1911.02685.

[27] "Introduction to the fourier transform." (), [Online]. Available: https://lpsa.swarthmore.edu/Fourier/Xforms/FXformIntro.html.

[28] C. Torrence and G. P. Compo. "A practical guide to wavelet analysis." (), [Online]. Available: https://lpsa.swarthmore.edu/Fourier/Xforms/FXformIntro.html.

[29] A. Makiewicz and W. Ratajczak, "Principal components analysis (pca)," Tech. Rep., 1993. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/009830049390090.

[30] A.-T. Dinh. "Principal component analysis (pca)." (), [Online]. Available: https://dinhanhthi.com/note/principal-component-analysis/.

A

Appendix 1: Prediction graphs

A.1 LSTM

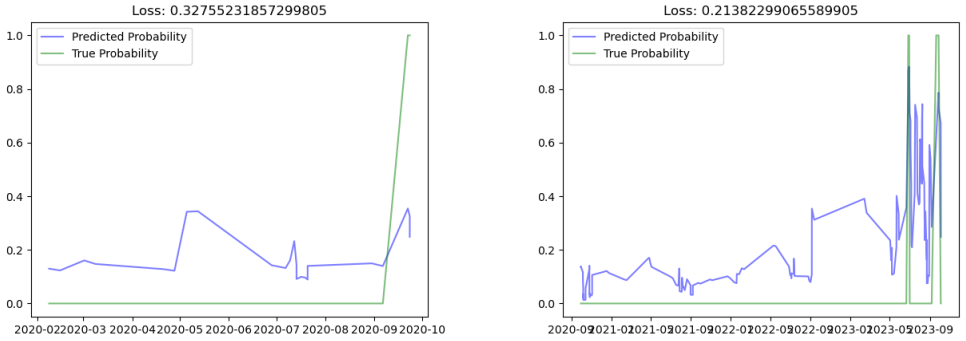

Figure A.1: The LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

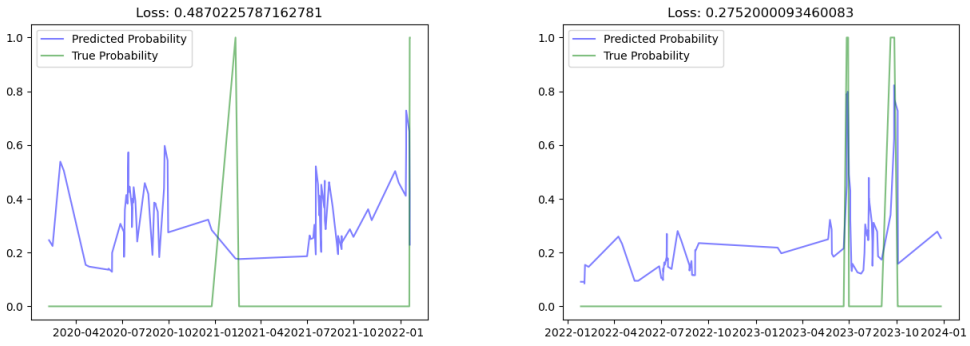

Figure A.2: The LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

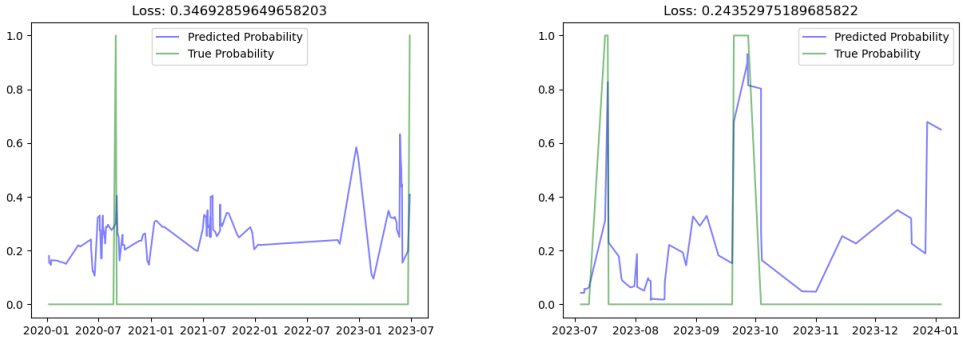

Figure A.3: The LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

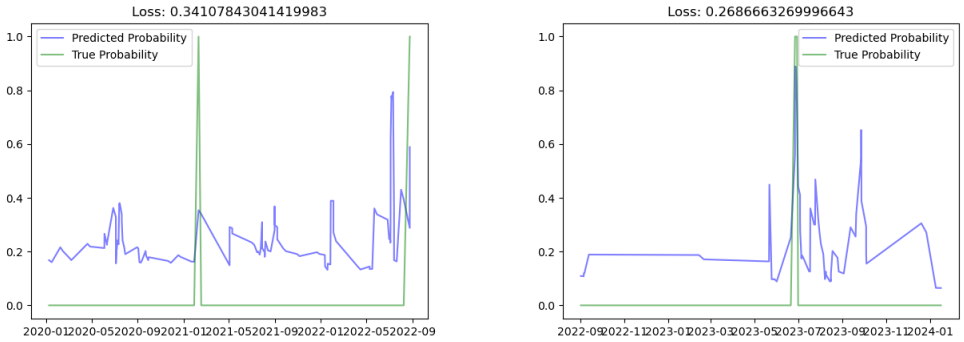

Figure A.4: The LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

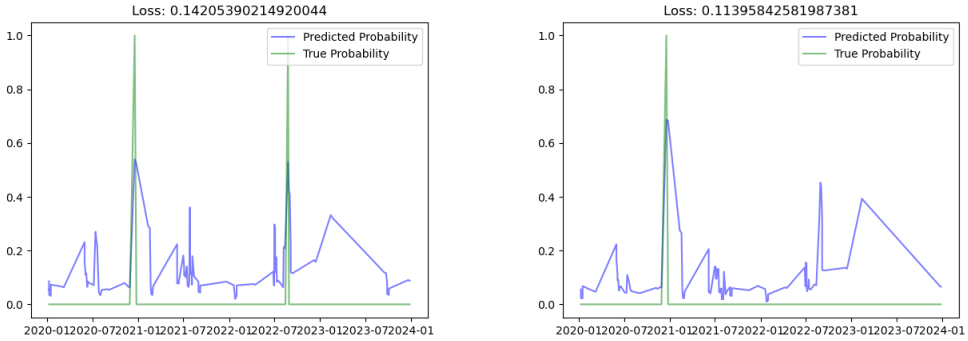

Figure A.5: The LSTM predicted probability (y-axis) over time (x-axis) of future interruption for two of the data locations with training data.

A.2 Transformer

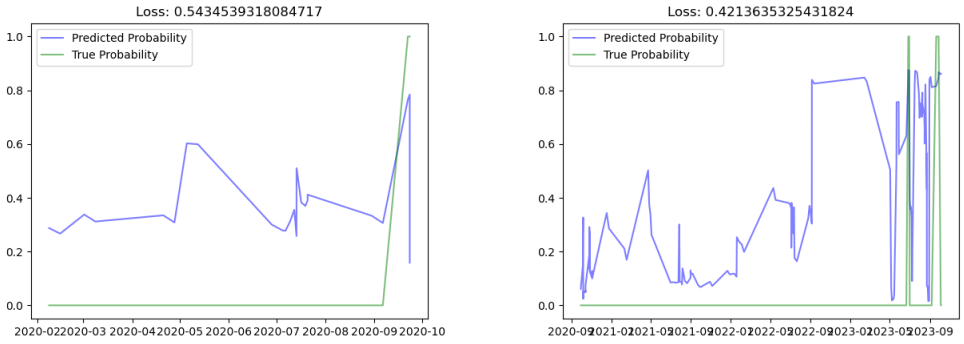

Figure A.6: The transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

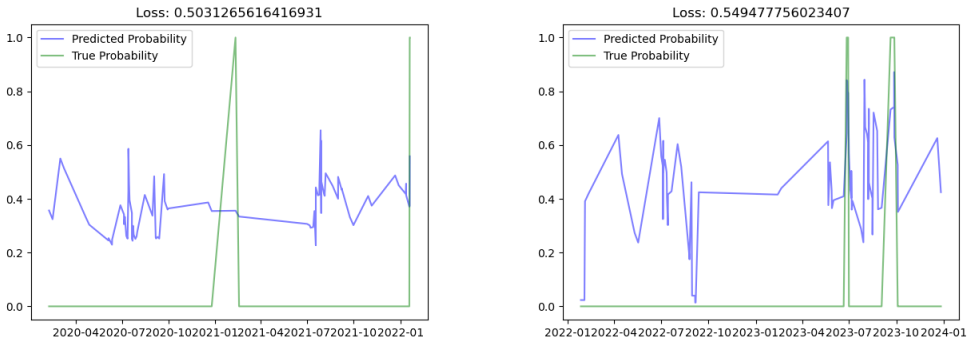

Figure A.7: The transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

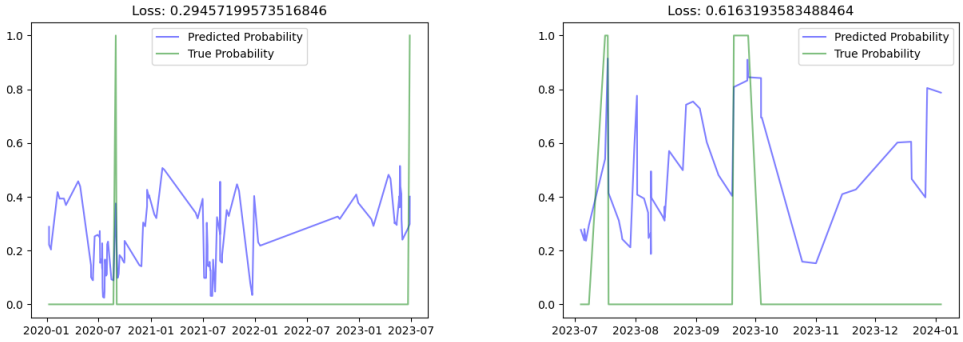

Figure A.8: The transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

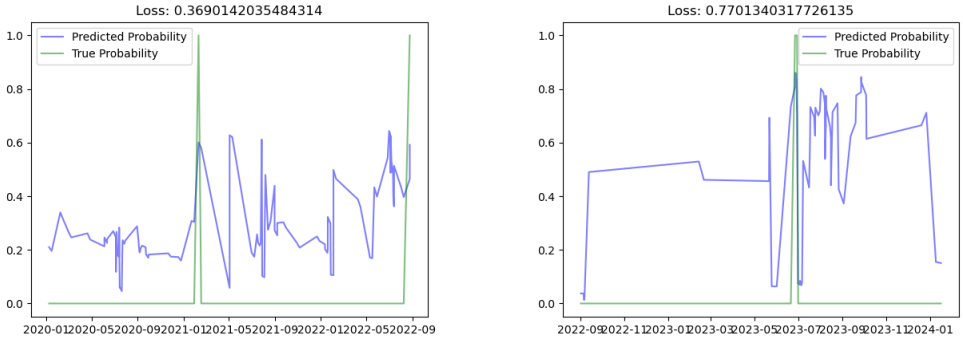

Figure A.9: The transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

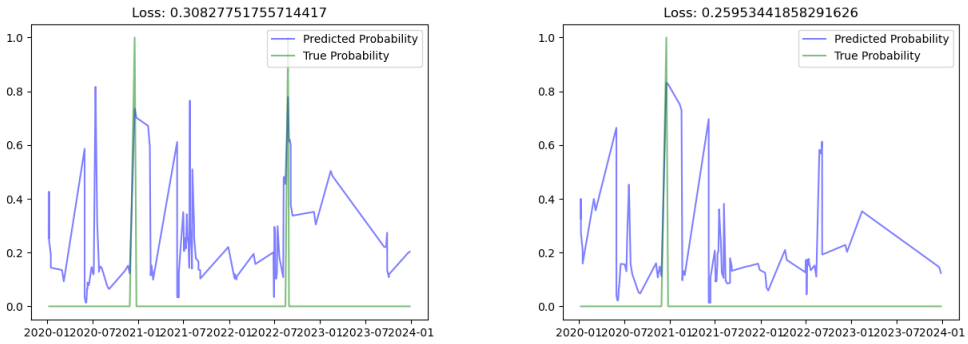

Figure A.10: The transformer predicted probability (y-axis) over time (x-axis) of future interruption for two of the data locations with training data.

A.3 LSTM (pre-trained)

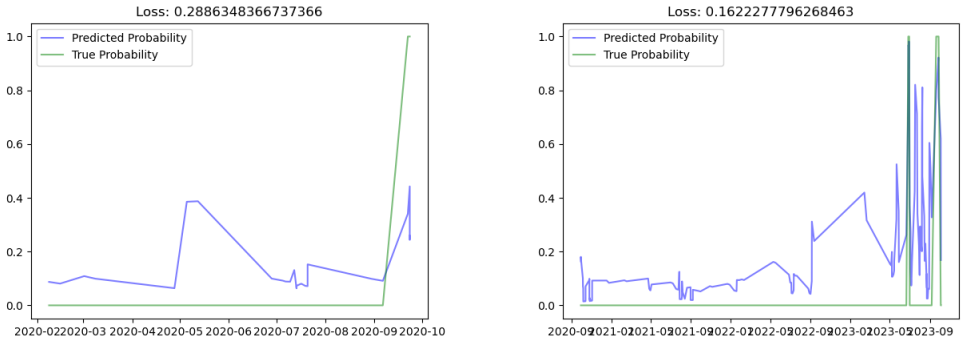

Figure A.11: The pre-trained LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

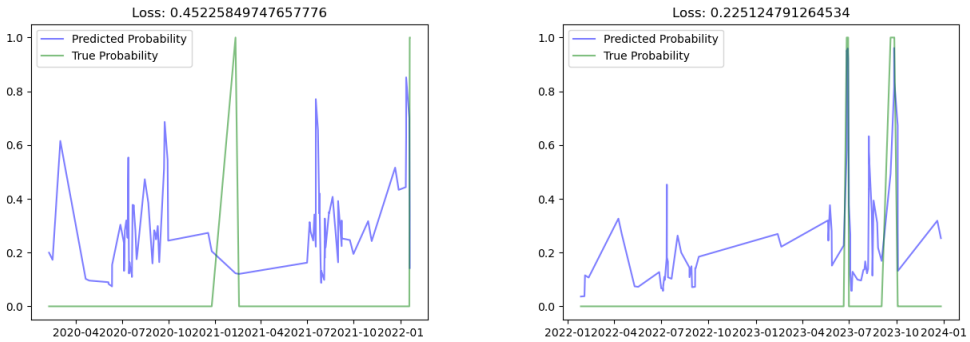

Figure A.12: The pre-trained LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

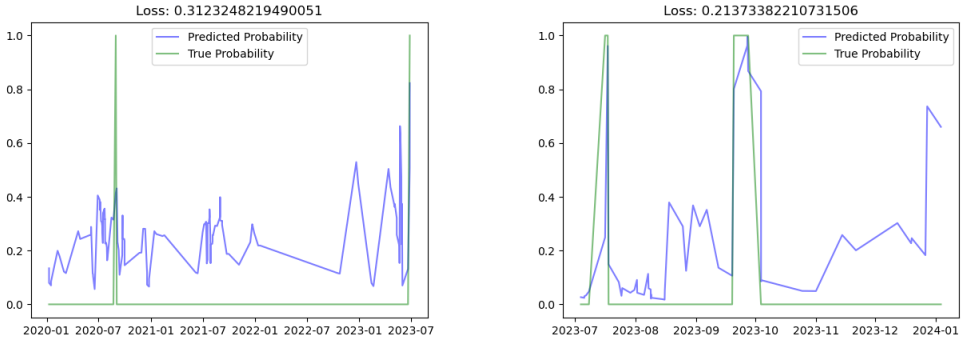

Figure A.13: The pre-trained LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

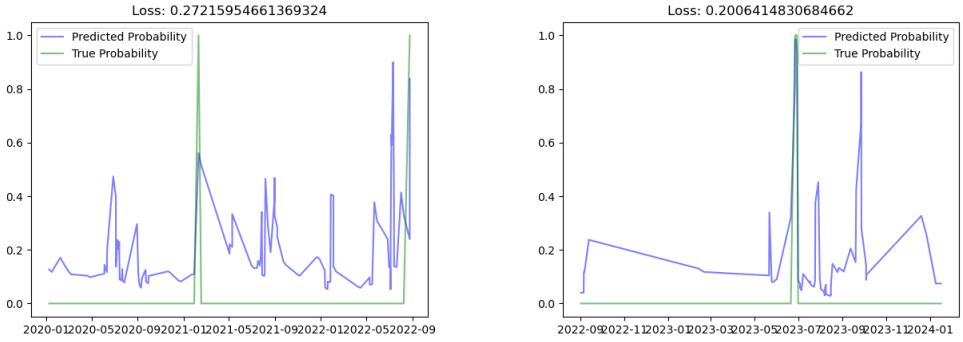

Figure A.14: The pre-trained LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

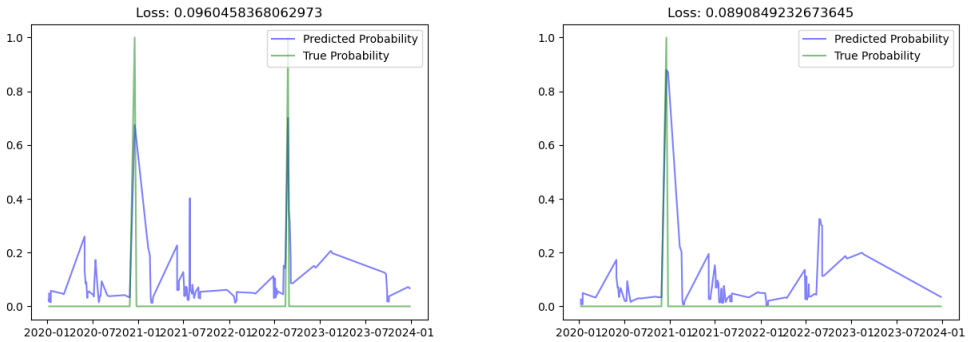

Figure A.15: The pre-trained LSTM predicted probability (y-axis) over time (x-axis) of future interruption for two of the data locations with training data.

A.4 Transformer (pre-trained)

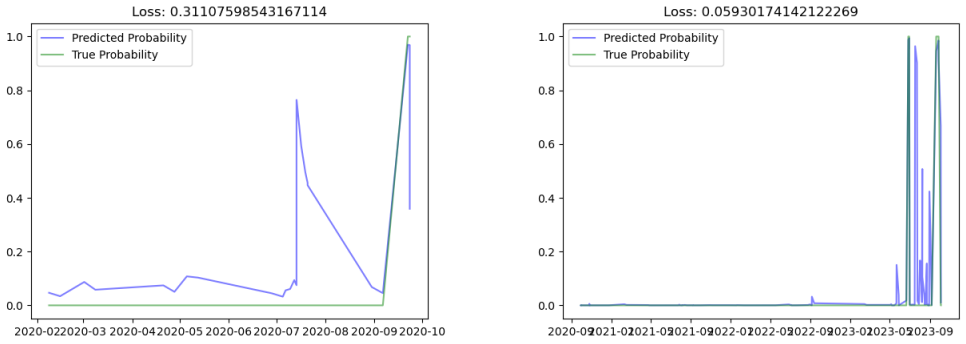

Figure A.16: The pre-trained transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

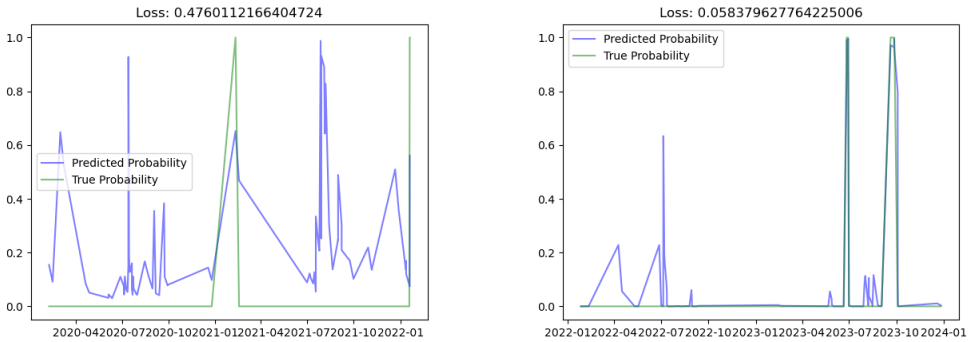

Figure A.17: The pre-trained transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

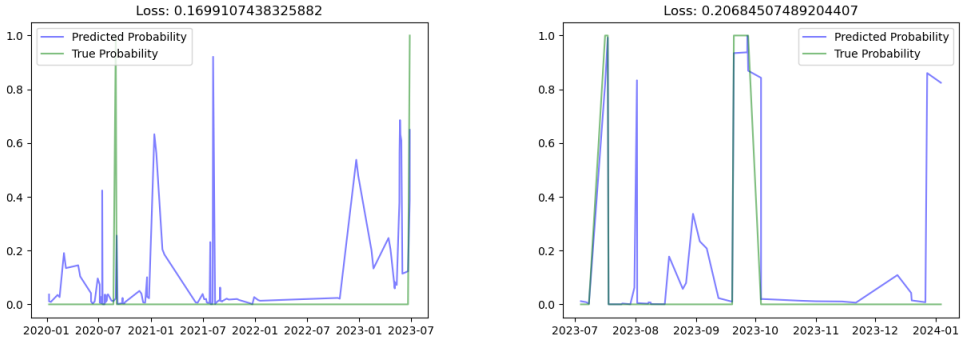

Figure A.18: The pre-trained transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

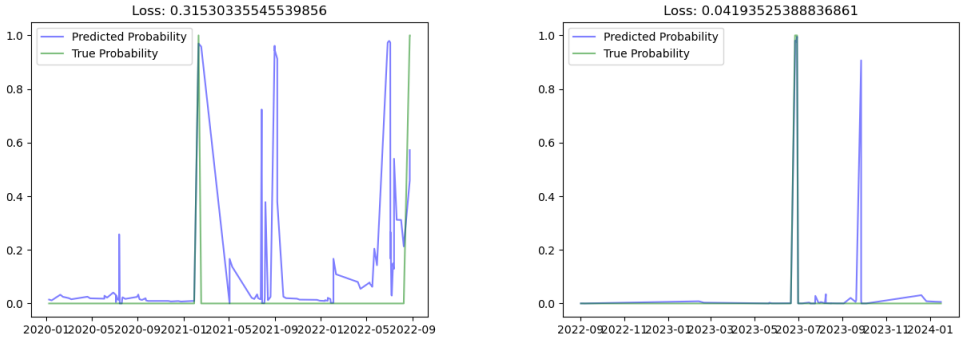

Figure A.19: The pre-trained transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

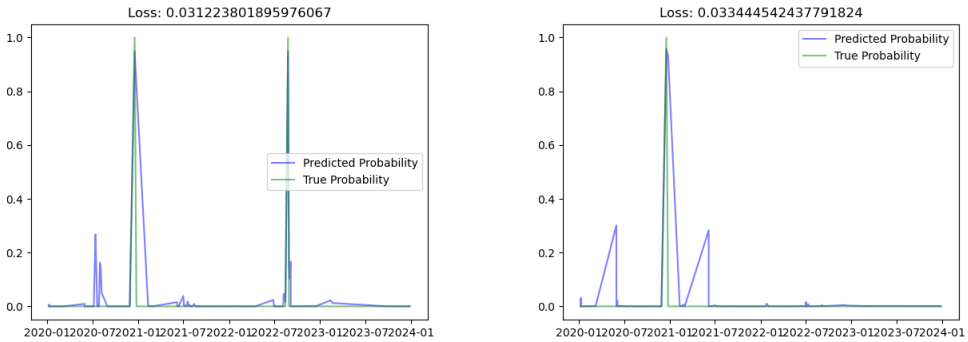

Figure A.20: The pre-trained transformer predicted probability (y-axis) over time (x-axis) of future interruption for two of the data locations with training data.

A.5 CNN LSTM

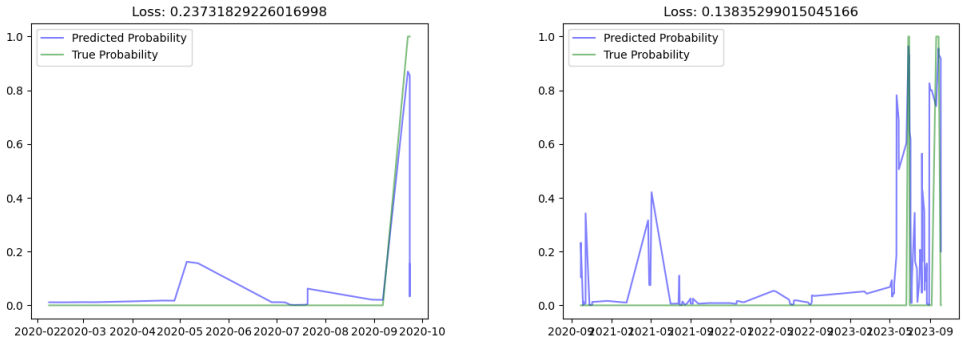

Figure A.21: The CNN LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

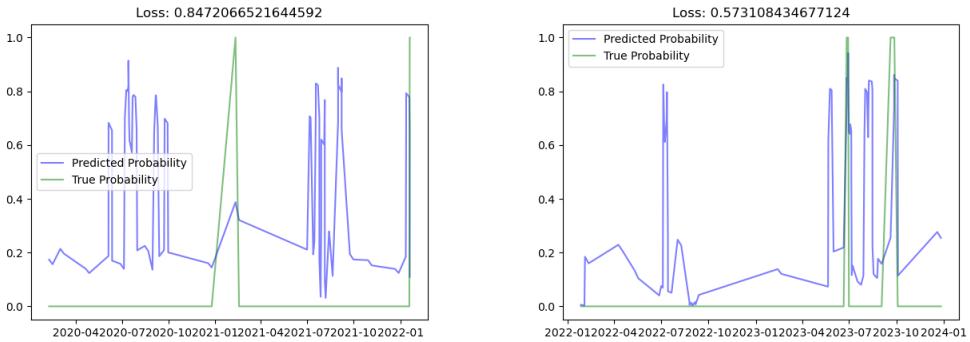

Figure A.22: The CNN LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

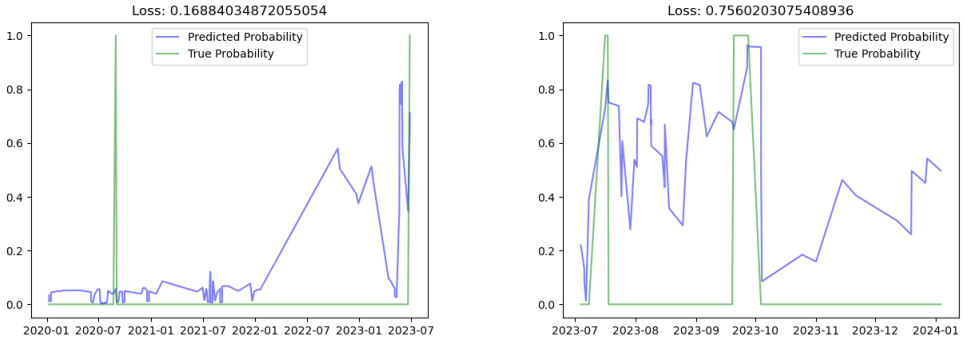

Figure A.23: The CNN LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

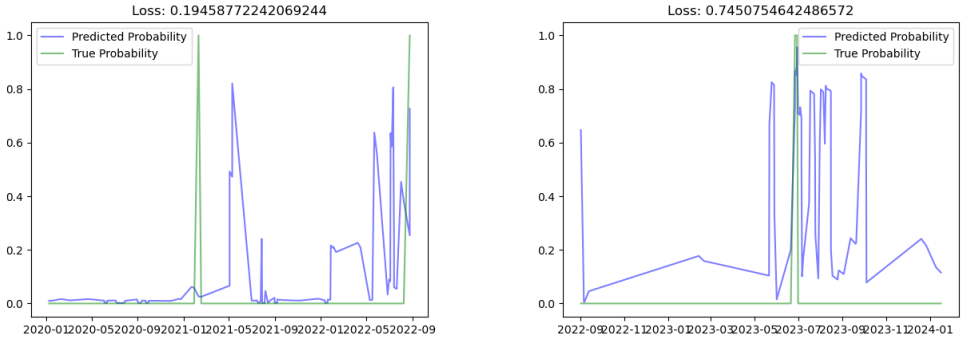

Figure A.24: The CNN LSTM predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

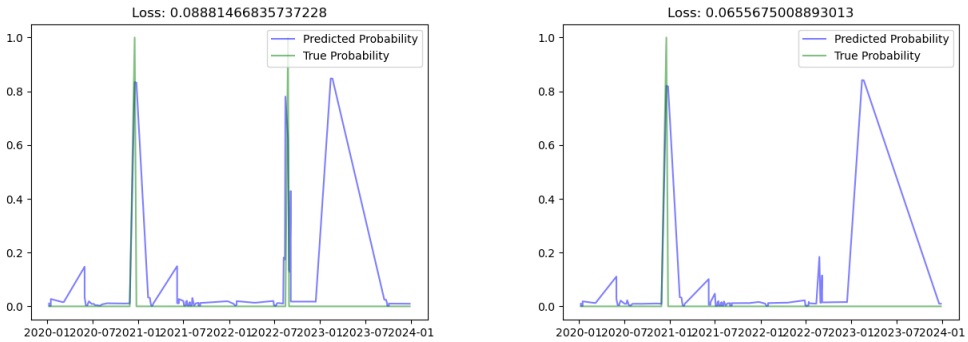

Figure A.25: The CNN LSTM predicted probability (y-axis) over time (x-axis) of future interruption for two of the data locations with training data.

A.6 CNN Transformer

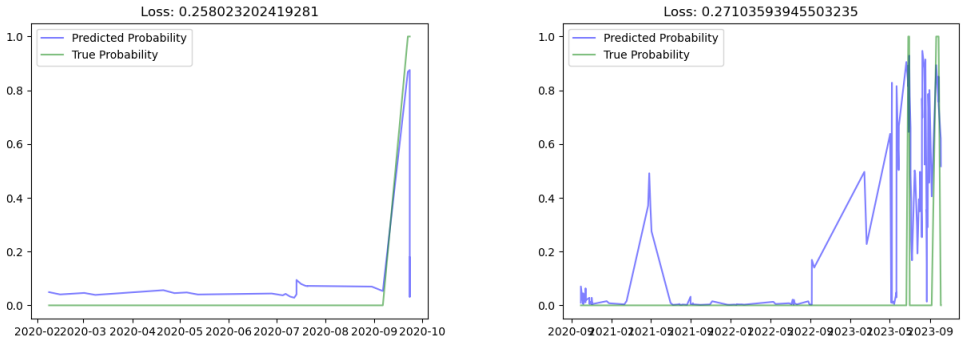

Figure A.26: The CNN transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

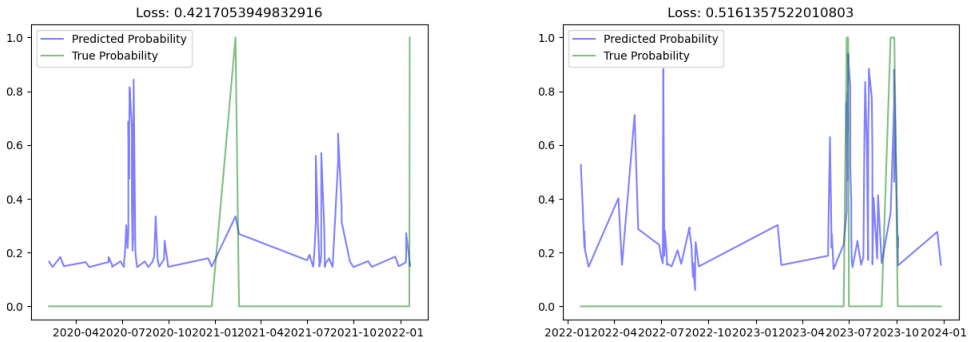

Figure A.27: The CNN transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

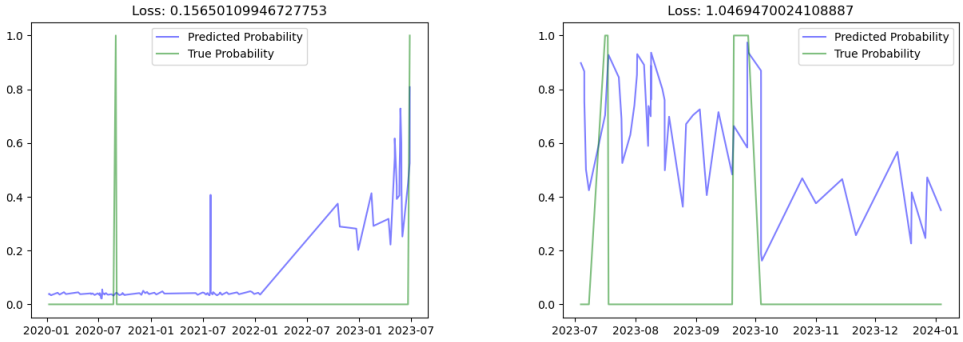

Figure A.28: The CNN transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

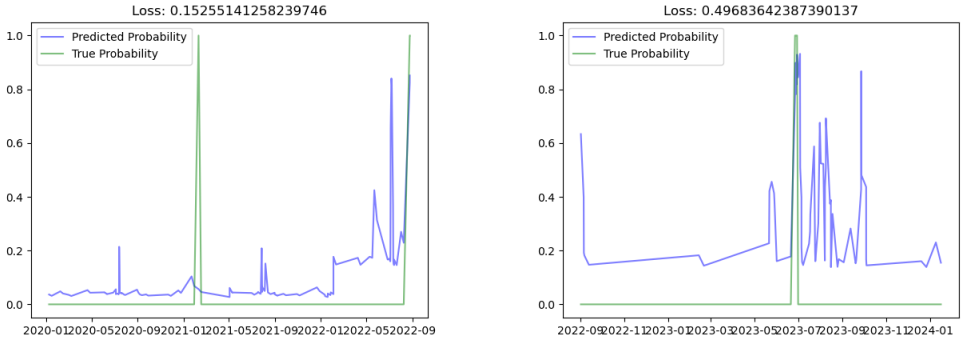

Figure A.29: The CNN transformer predicted probability (y-axis) over time (x-axis) of future interruption for one of the data locations. The left figure represents the predictions on the validation data while the right figure is the predictions on the training data.

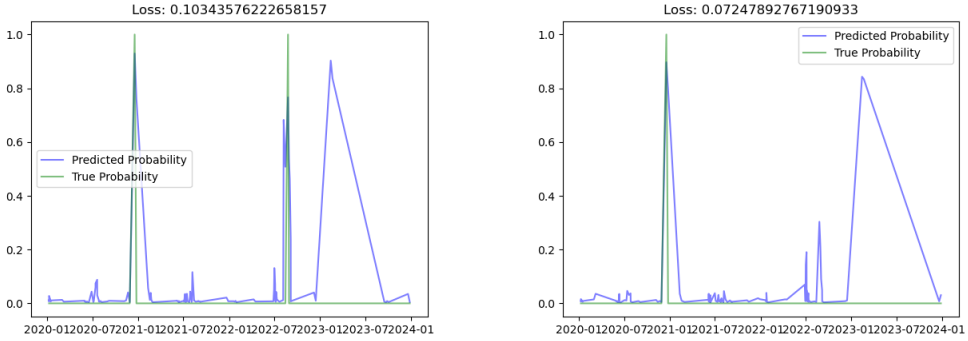

Figure A.30: The CNN transformer predicted probability (y-axis) over time (x-axis) of future interruption for two of the data locations with training data.