Warsaw University of Technology

FACULTY OF
MATHEMATICS AND INFORMATION SCIENCE

Master's diploma thesis

in the field of study Computer Science
and specialisation Data Science

Transfer learning for time series classification

Paulina Pacyna

student record book number 290600

thesis supervisor
Agnieszka Jastrzębska, PhD

WARSAW 2023

Abstract

Transfer learning for time series classification

The task of classifying time series is an important problem in the field of data mining. Most phenomena that change over time can be described in terms of time series. A time series can describe, for example, the amplitude of a heartbeat sound, stock prices, or hand movement along an axis when serving in tennis. Time series can express different characteristics of the phenomenon. Those characteristics are called *classes*. For example, the heartbeat sound amplitude can represent (belong to) one of two classes: *healthy* and *unhealthy*. Time series classification attempts to learn the distinctive features of a time series and build a model that can distinguish between those classes.

The time series classification problem was initially solved using classical algorithms such as the k-nearest neighbor classifier with distance measures suited for time series, like Dynamic Time Warping. The advantages of using deep learning algorithms in the context of time series classification have recently begun to be recognized. Neural networks can detect shapes that distinguish a class or understand ordered temporal relationships.

Transfer learning is practically used when there is limited data to train. Transfer learning attempts to apply patterns learned from one dataset to improve learning when creating a model for another dataset. A common practice is to prepare a source classifier trained on a large, readily available amount of data for one task and then use this model or parts of it for a detailed task with a smaller amount of data. Models trained using this method often have a shorter training time, faster accuracy increase, and can generalize more easily on the test set.

Keywords: time series, classification, transfer learning, deep learning, ...

Streszczenie

Zastosowanie techniki transfer learning w zadaniu klasyfikacji szeregów czasowych

Zadanie klasyfikacji szeregów czasowych jest ważnym problemem w dziedzinie eksploracji danych. Szeregi czasowe występują za każdym razem, gdy chcemy zmierzyć jakieś zjawisko, które zmienia się w czasie. Szereg czasowy może opisywać np. amplitudę dźwięku bicia serca, ceny akcji czy ruch ręki wzdłuż osi podczas serwu w tenisie. Takie szeregi czasowe mogą wyrażać różne cechy zjawiska. Te cechy nazywane są *klasami*. Na przykład amplituda dźwięku bicia serca może reprezentować (należeć do) jednej z dwóch klas: *norma* i *choroba*. Klasyfikacja szeregów czasowych polega na rozpoznawaniu charakterystycznych cech szeregu czasowego i budowaniu modelu, który potrafi rozróżniać klasy.

Problem klasyfikacji szeregów czasowych był początkowo rozwiązywany za pomocą klasycznych algorytmów, takich jak algorytm k-najbliższych sąsiadów w połączeniu z miarami podobieństwa dla szeregów, jak odległość DTW. W ostatnim czasie zaczęto dostrzegać zalety stosowania algorytmów głębokiego uczenia w kontekście klasyfikacji szeregów czasowych. Sieci neuronowe są w stanie wykryć kształty wyróżniające daną klasę lub zrozumieć relacje między obserwacjami w czasie.

Metoda transfer learning jest stosowana w przypadku ograniczonej ilości danych do trenowania modelu. Technika transfer learning próbuje zastosować wzorce wyuczone z jednego zbioru danych, podczas tworzenia modelu dla innego zbioru danych. Częstą praktyką jest przygotowanie źródłowego klasyfikatora wytrenowanego na dużej, łatwo dostępnej ilości danych dla jednego zadania, a następnie wykorzystanie modelu lub jego części do szczegółowego zadania z mniejszą ilością danych. Modele wytrenowane tą metodą często mają krótszy czas trenowania, szybszy wzrost dokładności i lepsze, ogólniejsze wyniki na zbiorze testowym.

Słowa kluczowe: szeregi czasowe, klasyfikacja, transfer learning ...

Contents

1. Introduction

Time series classification was initially approached using classical machine learning algorithms. Bagnall et al. in [1] categorizes the commonly used algorithms into several categories. The first category is time domain distance-based algorithms. Those algorithms use various distance measures adjusted for the time series domain to capture the similarity between pairs of time series. The distance measure is then combined with a distance-based classifier. A flagship example of an algorithm belonging to this class of algorithms is Dynamic Time Warping distance with the k-Nearest Neighbour classifier. The DTW-1-NN classifier is often used as a benchmark classifier. Another category of classifiers mentioned in [1] are dictionary, shapelet and interval based classifiers. All try to extract distinctive features for the time series class. The dictionary-based classifiers encode the time series into a dictionary of *words* representing the time series. Shapelet-based algorithms focus on subseries of the time series that are discriminatory of class membership. Interval-based algorithms extract features from selected intevals of the time series.

With the rise of the popularity of deep learning algorithms, researchers attempted to replace the former, hand-extracted features and algorithms with deep learning classifiers. Fawaz et al. reviewed deep learning algorithms applied to the time series classification task [7]. The usage of deep learning algorithms enabled the possibility to utilize transfer learning.

Transfer learning is widely used in image recognition and natural language processing. Fawaz et al. extended their previous findings by a study on the application of transfer learning [7]. The authors examine knowledge transferability on 85 datasets from the UCR archive by pre-training a model on one dataset and fine-tuning it on another. The authors examine if the accuracy improved for all pairs of datasets in the UCR archive.

Transfer learning for deep learning is usually done by training the network on a big, diverse dataset and utilizing the first layers of the network when training on another dataset. In image recognition or natural language processing, it is common to pre-train the source network on the ImageNet dataset or Wikipedia dataset, respectively. Such a diverse dataset with labels does not exist for time series. In this thesis, we will attempt to create an artificial, diverse dataset from 85 smaller datasets available on the UCR archive. We will try to mimic the good properties of a transfer learning source dataset by preprocessing, upsampling, and augmenting the dataset. We will also study if transfer learning on a diverse dataset helps to generalize new training data and improve the training process on the target dataset. Similarly, as in [8], we will experiment to find which features of the source dataset (classes diversity, dataset size, augmentation, preprocessing) are fundamental for the transfer learning process.

2. Related works

In this chapter, we describe several algorithms used in time series classification, focusing on deep learning algorithms. We will refer to literature and papers in the domain of time series classification and transfer learning. We will also describe the most popular archive of time series datasets, the UCR archive.

2.1. Time series classification

A time series is an ordered collection of observations indexed by time.

$$X = (x_t)_{t \in T} = (x_1, ..., x_T),\ x_t \in \mathbb{R}$$

The time index T can represent any collection with the natural order. We assume that indices are spaced evenly in the set T. The realization or observation x_t in the times series is a numerical value describing the phenomena we observe, for example, the amplitude of a sound, stock price, or y-coordinate. Time series classification is a problem of finding the optimal mapping between a set of time series and corresponding classes.

2.1.1. Multi Layer Perceptron

The Multi Layer Perceptron (MLP) is the first artificial neural network architecture proposed in [7] and can be used for time series classification tasks. Formally, the MLP network can be defined as a composition of functions (called *dense layers* or *layers*). The network returns a vector that usually represents the probability distribution over the set of classes.

$$MLP(X; \theta_1, \ldots, \theta_M, \beta_1, \ldots, \beta_M) = L_M(\ldots L_2(L_1(X; \theta_1, \beta_1); \theta_2, \beta_2); \theta_M, \beta_M)$$

Each layer $L_i : \mathbb{R}^M \to \mathbb{R}^N$ is a function that depends on the parameters $\theta \in r^{M \times N}, \beta \in \mathbb{R}^N$

$$L_i(X; \theta_i, \beta_i) = f_i(X\theta_i + \beta_i)$$

The function $f_i : \mathbb{R}^N \to \mathbb{R}^N$ is an arbitrarily chosen non-linear function. The number of layers and dimension of weights in hidden layers is also arbitrary. The weights in the first and last

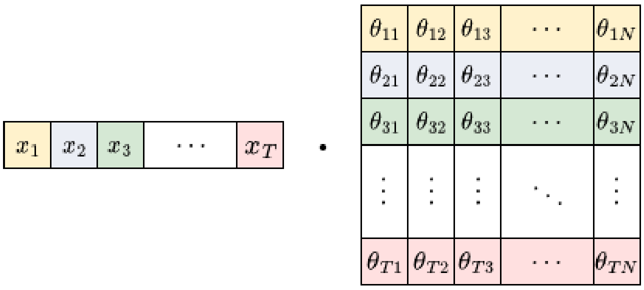

Figure 2.1: The multi layer perceptron architecture treats each entry in the input time series separately.

layer have to match the dimensionality of input data (e.g. the length of the time series) and the number of classes. The output of the last layer is interpreted as a probability distribution over the set of classes.

The disadvantage of using Multi Layer Perceptrons for time series classification is that the input size is fixed. All time series in the training data must have the same length. In transfer learning, this means that if we want to reuse the source network (or a set of first layers from the network), the time series in the target dataset must have the same length as in the source dataset.

The MLP architecture fails at understanding the temporal dependencies [7]. Each input value in the time series is treated separately because it is multiplied by a separate row in the weight matrix (see figure 2.1.1 for an illustration).

2.1.2. Convolutional Neural Networks

Convolutional Neural Networks are widely used in image recognition. A convolution applied for a time series can be interpreted as sliding a filter over the time series. A convolutional layer is a set of functions called convolutions or filters. The filter is applied at a given point, using the values surrounding the point.

To define the convolution operation, let's assume the input is a matrix $X \in \mathbb{R}^{(N_1,\ldots,N_K)}$. In the case of images, the number of dimensions K is often equal to 3 (height, width, channels), for univariate time series we can assume just one dimension, and for multivariate time series we need two dimensions - (feature, time). The filter consists of a matrix of weights $M \in \mathbb{R}^{(P_1,\ldots,P_K)}$. Usually, P_l are odd numbers, so that we can index the matrix with symmetrical numbers: $(\frac{-P_l+1}{2}, \frac{-P_l+3}{2}, \ldots, 0, \ldots, \frac{P_l-1}{2})$. The 0 index marks the center of the matrix.

Finally the convolution $*$ is defined as follows:

$$(X * M)_{i_1,\dots,i_K} = \sum_{l_1=\frac{-P_1+1}{2}}^{\frac{P_1-1}{2}} \cdots \sum_{l_K=\frac{-P_K+1}{2}}^{\frac{P_K-1}{2}} M_{l_1,\dots,l_K} X_{i_1+l_1,\dots,i_K+l_K}$$

The result of the convolution is passed elementwise to a nonlinear function. The nonlinear function, together with the convolution operation, will be called a filter. In the case of univariate time series, the first layer of the convolutional neural network is one-dimensional. The output of the first layer has dimensions (length of time series - the length of the filter + 1, number of filters). Below we define the value of the output for filter i

$$y_{t,i} = f_i([\theta^i_{\frac{-M+1}{2}}, \dots, \theta^i_{\frac{M-1}{2}}] \cdot [X_{t+\frac{-M+1}{2}}, \dots, X_{t+\frac{M-1}{2}}]),$$

where $\cdot$ is a dot product and $t \in T$ is the time index. In figures 2.2 and 2.3 we show the results of applying two different filters to a time series.

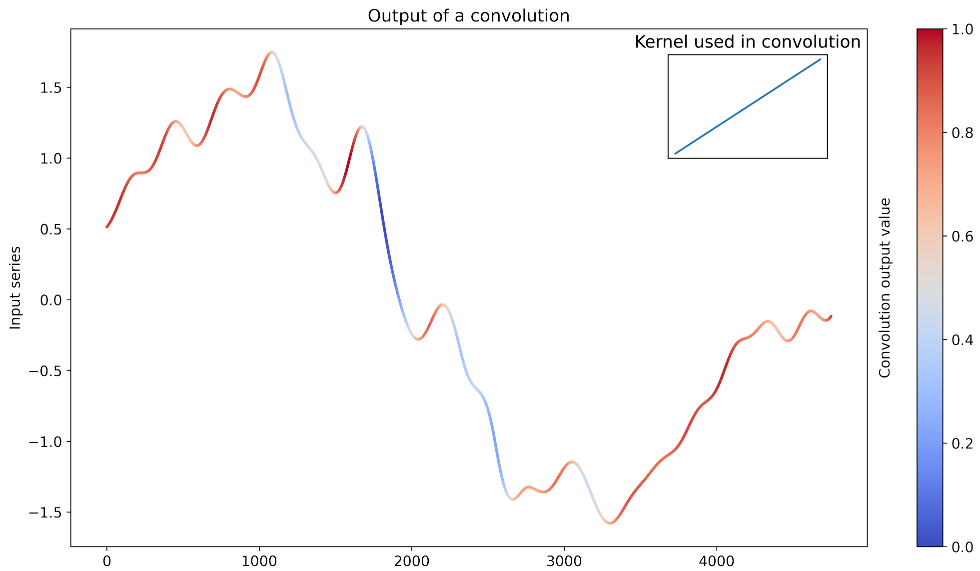

Figure 2.2: Result of applying a "slope" detecting filter to a time series. Red regions indicate a higher value of the output convolution.

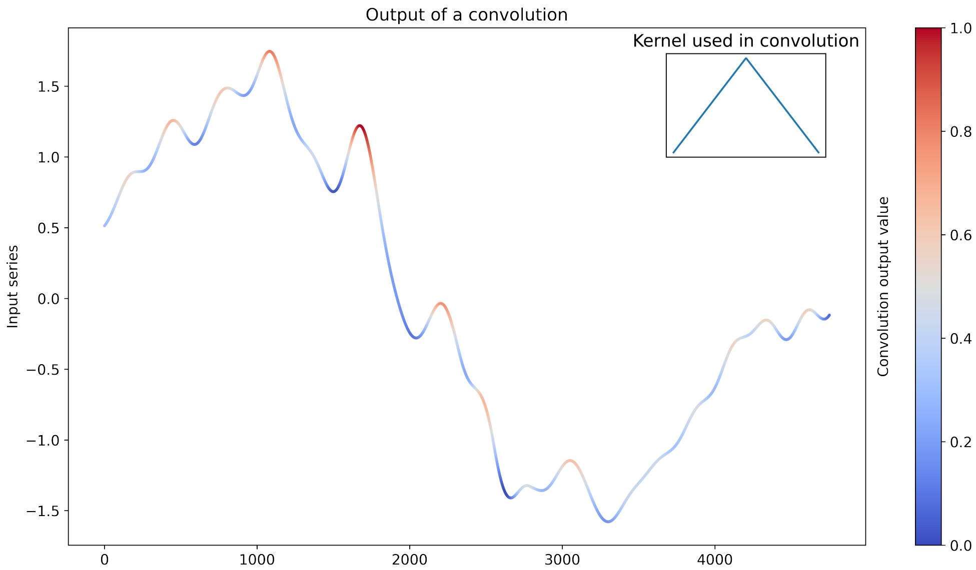

Figure 2.3: Result of applying a "peak" detecting filter to a time series. Red regions indicate a higher value of the output convolution.

The weights θ^i are different for each filter. The same filter is applied over the whole length of the time series. This is called *weight sharing*, enabling the network to learn patterns regardless of the position in the time series.

The architecture of the convolutional layer is not dependent on the input data size. Regardless of the size of the input data, the number of filters and size of filters remain the same, and only the output sizes depend on the input size. Therefore, if the convolutional layer is succeeded by layers with the same property, like other convolutional layers or Global Pooling with Dense Layer (see Section 2.1.3), the whole network may be invariant to the input sizes [7]. Such networks may be interesting in transfer learning, as the sizes of time series in the source and target tasks do not have to match.

2.1.3. Fully Convolutional Networks

Fully Convolutional Networks are convolutional networks used in time series classification. A sample architecture of a Fully Convolutional Network proposed is in [7]. The first layers in the network are 3 blocks of convolutional layers with ReLU activation function followed by batch normalization layers. The output of the last block is passed to a global pooling layer. The global pooling layer averages the output through the time axis, resulting in a vector of length equal to the number of feature maps in the last convolutional layer. The averaged vector is passed to a block of 2 fully connected layers. Figure 2.4 shows a visualization of the network.

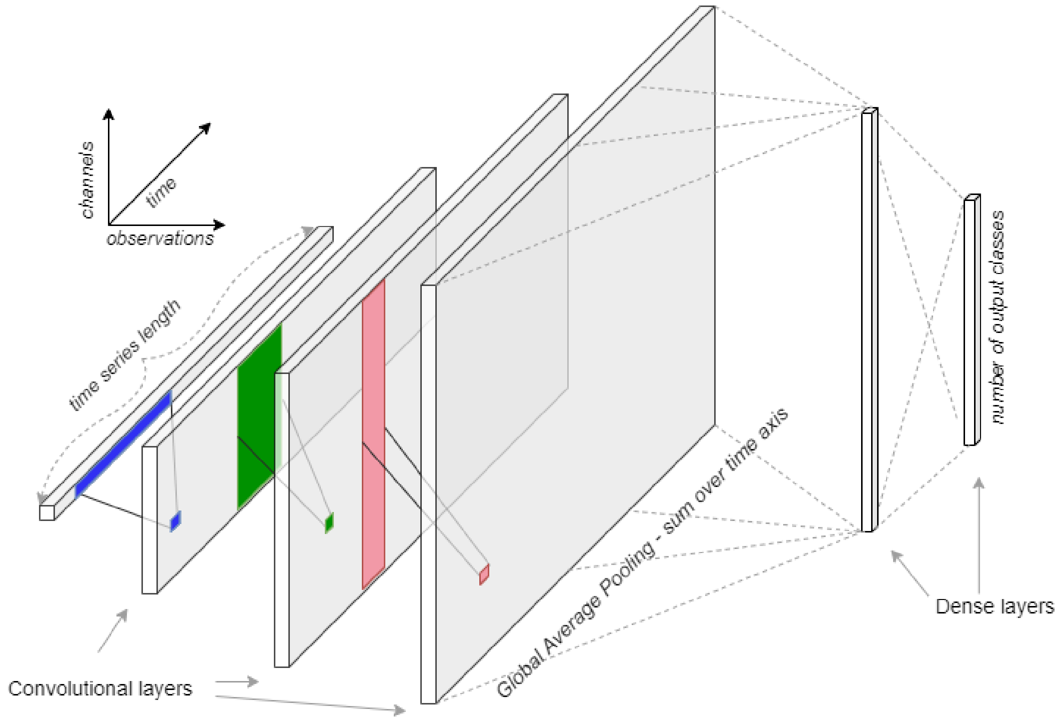

Figure 2.4: Architecture of a Fully Convolutional Network. Source: [7]

Because the architecture of convolutional layers does not depend on the size of input data and the convolutional layers are followed by pooling over the time axis, the whole network is capable of processing data of variable lengths.

2.1.4. Residual Network

The Residual Network was first proposed in [13]. The Residual Network is a relatively deep architecture compared to other neural networks used for time series classification.

A residual connection addresses the vanishing gradient problem occurring in networks composed of many layers [13]. The vanishing gradient problem occurs in the backpropagation process. When the gradient passes from the last layer to the first, it may decrease toward zero. This causes the first layers of the network to learn slowly. The residual connection between layers passes the input directly from one layer to another, skipping a few layers. This way, if the network is struggling with a vanishing gradient, this shortcut connection may help the network to converge.

The architecture of the Residual Network is conceptually similar to the FCN network. Instead of three convolutional layers, the Residual Network begins with three blocks of convolutional layers, connected with residual connections. Each block consists of three convolutional layers. The three blocks, which consist of nine convolutional layers and three residual connections in total, are followed by a Global Average Pooling layer and a Dense layer. The networks' diagram is shown below (Figure 2.5).

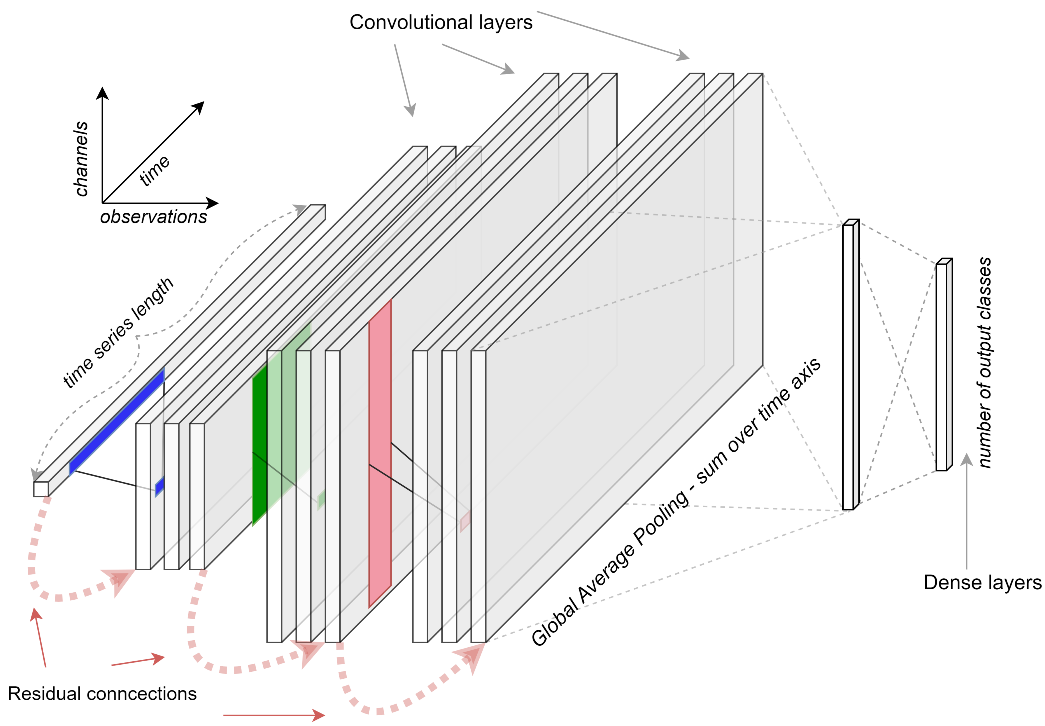

Figure 2.5: Architecture of a Residual Network. Source: [7]

2.1.5. Encoder Network

The Encoder Network is a deep convolutional network proposed by [11]. The first layers of the network are similar to the FCN architecture. The network consists of three convolutional layers followed by an attention layer instead of the Global Average Pooling layer. Another novelty introduced in this network compared with the FCN architecture is adding a DropOut layer and replacing the ReLU activation function with PReLU (Parametrized ReLU), thus adding another parameter to the network. The network ends with a Dense layer predicting the distribution over the classes. The architecture is shown below (Figure 2.6).

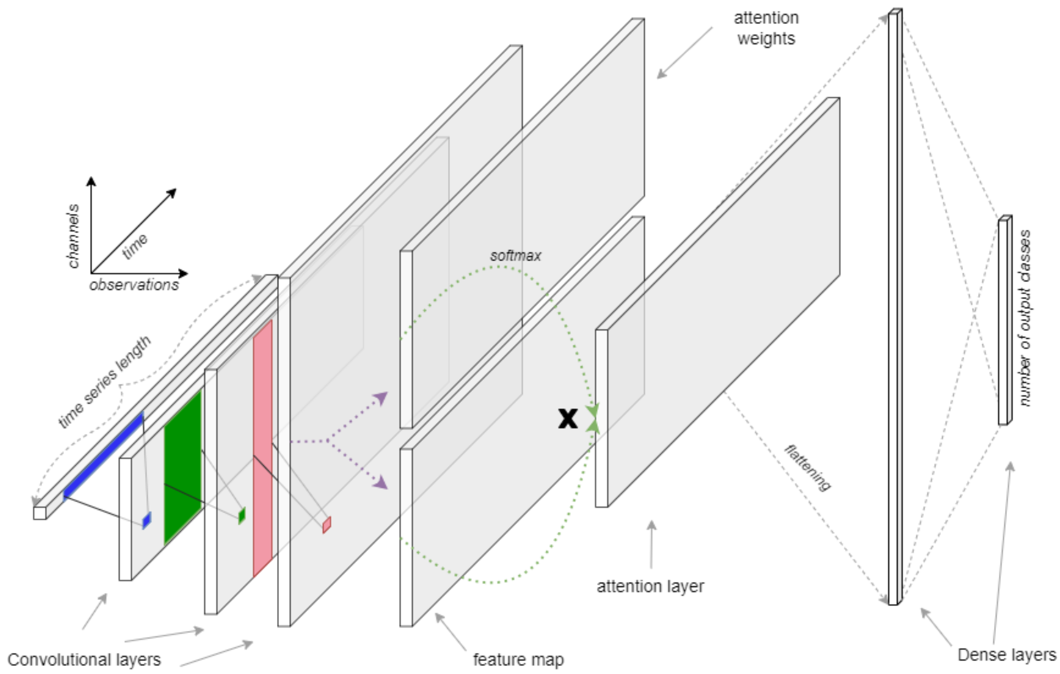

Figure 2.6: Architecture of an Encoder Network. Source: [7]

The attention layer, first proposed in [2], assigns weights to each input element representing the importance of the prediction. The weights representing the attention are normalized using the softmax function and then applied to the input data. The weights are learned by the network. The outputs of the attention layer can be interpreted as a universal representation of the input time series that can adapt to and represent unseen data [11].

As opposed to the FCN network architecture, the Encoder network is not invariant to the input size. The Dense layer parameters depend on the output size of the attention layer and implicitly on the output size of convolutional layers. Similarly, as in the former architecture, the convolutional layers enable weight sharing, thus learning shapelets independently of their position in the time series.

2.2. The UCR archive

The URC (University of California, Riverside) archive is a time series classification archive introduced in 2002 by Dau, Keogh et al. [5]. At moment of writing this thesis, it consists of 128 univariate time series datasets. The time series describes various phenomena, like readings from a device or sensor, motion recordings, or food spectrographs. The summary of the time series types is listed below (Table 2.1). We also show a few time series samples from different datasets below (Figure 2.7).

Dataset type (source)	Number of datasets
Device	9
ECG (Electrocardiogram)	6
EOG (Electrooculography)	2
EPG (Electrical penetration graph)	2
Hemodynamics	3
HRM (High Resolution Melt)	1
Image	32
Motion	17
Power	1
Sensor	30
Simulated	8
Spectro	8
Spectrum	4
Traffic	2
Trajectory	3

Table 2.1: Summary of types of datasets in the UCR archive.

All datasets together contain 1118 unique labels. Most of the datasets differentiate between two labels and have several dozen rows per class. The series length varies from 8 to almost 3 thousand. Below we show four histograms summarizing the number of classes per dataset, the lengths of the series, and the number of observations per class and per dataset. (Figure 2.8) The datasets in the archive are already split into train and test datasets. Most of the time series are already z-normalized. However, the authors recommend testing on additional test/train splits and preprocessing the time series accordingly to the method used.

The authors of [4] composed a set of best practices for the dataset users when conducting experiments on the time series archive. The first recommendation is to avoid *cherry-picking*, which tests the method on only a subset of the time series. It is recommended to report the results on all datasets. The second recommendation is to tune hyperparameters using only the train dataset. Another recommendation is to avoid claiming that the improvement is caused by a particular feature of the algorithm without extensive tests of the feature. To illustrate why it is crucial, the authors bring up an example of attaching accuracy improvement in classifiers based on wavelet transform to the multi-resolution property of the transformation. The authors show that in some cases, the accuracy gain is caused by the fact that the wavelet transform is implicitly smoothing the time series, thus the classifier could be improved by simply smoothing the time series. The authors recommend checking whether removing the feature worsens the

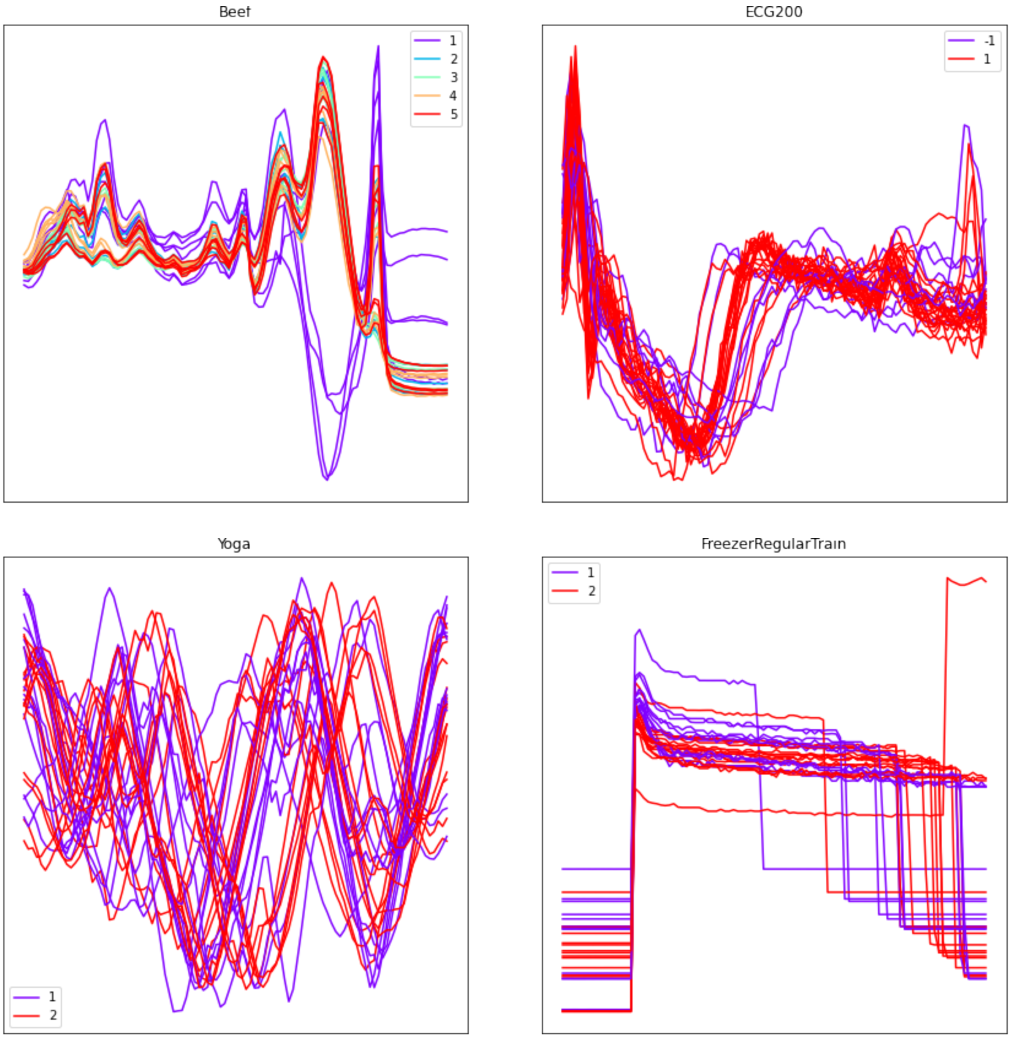

Figure 2.7: Example time series from the UCR archive. The images belong to the following categories: Food, ECG, Image, Device.

performance or comparing it against a different method but with the same property.

2.3. Preprocessing and augmenting time series

Data preprocessing is the process of preparing, cleansing, and manipulating the data to optimize the training process. Data augmentation is a process used to increase the amount and variability of data. Data augmentation is beneficial when there is a limited amount of data, as it can prevent overfitting.

Many classical and deep learning models were invented and designed together with the data preprocessing and augmenting step [1, 7]. One such method is Window Slicing [7]. This method extracts subsequences of the original series. The resulting subsequences are then concatenenated

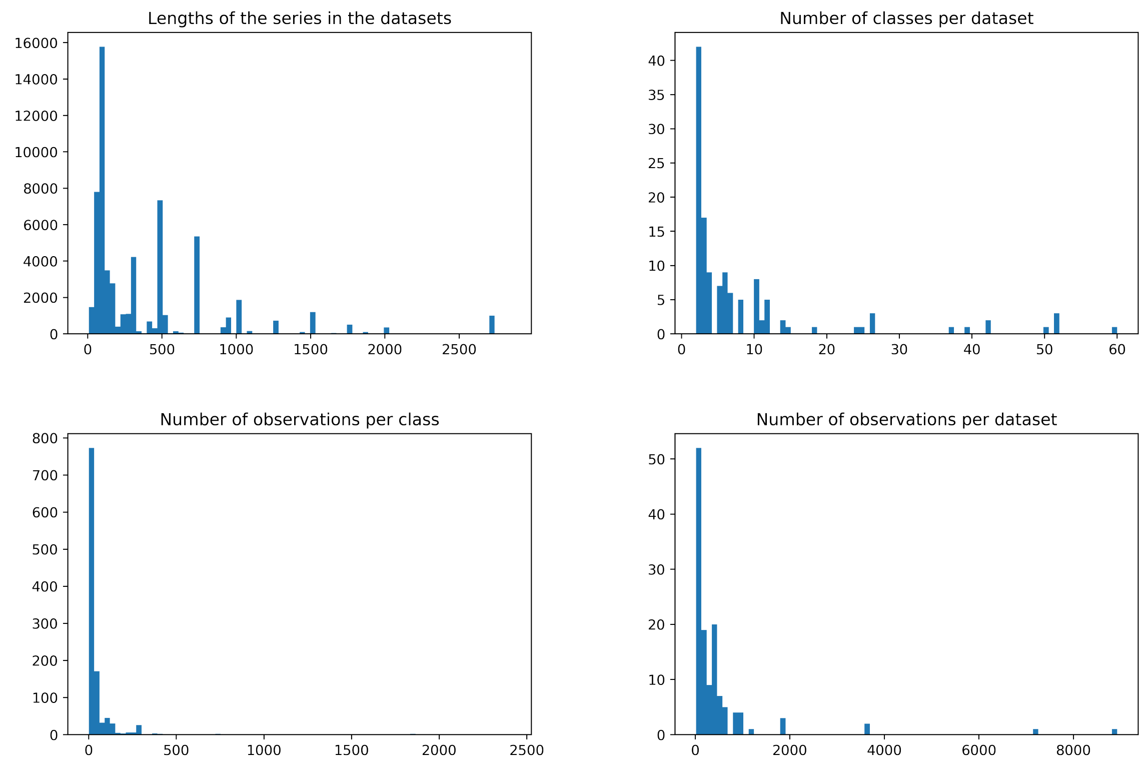

Figure 2.8: Summary of the UCR archive. The first plot is a histogram of time series length. The second histogram shows the number of classes in datasets. The two last histograms summarize the number of rows per class and per dataset.

with a downsampled copy and smoothed copy. This method is used in Multi-scale Convolutional Neural Network [3] and Time Le-Net [9].

Another preprocessing and augmentation method proposed together with the former method in the Time Le-Net model was Window Warping. It aims at making the model more robust to perturbations in the time axis by training the network on *squeezed* or *dilated* series together with the original series. The *dilated*/*squeezed* time series is two times longer/shorter than the original time series. The dilated, squeezed, and original time series are concatenated, forming a vector 3.5 longer than the original time series. In the Time Le-Net model, this vector undergoes the Window Slicing preprocessing step before training.

Data augmentation may also be achieved as a by-product of Generative Adversarial Networks. Generative Adversarial Networks Are composed of two components - discriminator and generator. The latter generates synthetic data and the discriminator predicts if the data is artificial or real. Results from the trained generator can be used as an augmented time series. TimeGAN network is an example of a Generative Adversarial Network suitable for processing time series [15].

2.4. Transfer learning

Transfer learning is a technique that attempts to apply knowledge learned while solving one task to enhance the learning process for another task. Formally, the problem can be described using the notions of tasks and domains [14, 16]. A *Domain* is a pair $\mathcal{D} = (\mathcal{X}, P(\mathcal{X}))$, where $\mathcal{X}$ is the feature space (e.g., the time series observations, and $P(\mathcal{X})$ is the probability distribution over the feature space. A *Task* is a pair of label space $\mathcal{Y}$ and the decision function f, $\mathcal{T} = (\mathcal{Y}, f)$. The decision function f is learned from $\mathcal{X}, P(\mathcal{X}), \mathcal{Y}$ in the learning process.

Transfer learning attempts to utilize knowledge from different domain/domains and task/-tasks. Formally, given $S \in \mathbb{N}$ source domains and source tasks $(\{(\mathcal{D}_i^S, \mathcal{T}_i^S) : i = 1, \ldots, S\})$ and $T \in \mathbb{N}$ target domains and target tasks $(\{(\mathcal{D}_i^T, \mathcal{T}_i^T) : i = 1, \ldots, S\})$ transfer learning utilizes knowledge learned from source domains and tasks to improve the learning process of decision functions in target tasks $\mathcal{T}_i^T$

2.4.1. Transfer learning categorization

Transfer learning can be categorized from different points of view [16]. One of the ways in which we can divide transfer learning algorithms is based on the availability of the labels:

- **inductive** transfer learning - labels are available for both source and target datasets
- **transductive** transfer learning - labels are available only in the domain dataset
- **unsupervised** transfer learning - no labels available in either dataset

Another way of dividing transfer learning methods is by comparing feature space distribution and labels. If the source and training dataset consists of the same features, belonging to a similar distribution $(\mathcal{D}^{\mathcal{S}} = (\mathcal{X}, \mathcal{P}(\mathcal{X}))^{\mathcal{S}} = (\mathcal{X}, \mathcal{P}(\mathcal{X}))^{\mathcal{T}} = \mathcal{D}^{\mathcal{T}}))$ and the label spaces are equal $(\mathcal{Y}^{\mathcal{S}} = \mathcal{Y}^{\mathcal{T}})$, then the task can be described as homogeneous transfer learning. If at least one of the former assumptions does not hold, the transfer learning setting is heterogeneous.

Transfer learning can be categorized based on the algorithm/approach used. There are four main categories [12] used in deep learning:

- **instance** based transfer learning - target dataset is enriched by instances from the source dataset belonging to the target distribution.
- **mapping** based transfer learning - source and target dataset are mapped to one domain. The distribution is adjusted in both datasets.

- **network** based transfer learning - target classifier uses parts of the network from the source classifier f.
- **adversarial** based transfer learning - an adversarial network tries to distinguish between the two datasets. If the network fails at this task, then it means that the extracted features are similar between source and target datasets.

2.4.2. Characteristics of a good source domain

In the field of image processing, it is widespread to use convolutional neural networks pre-trained with the ImageNet dataset [8]. ImageNet is a large dataset of human-annotated images. It contains 1 million labeled images of 1000 classes. The label space consists of fine-grained classes such as breeds of dogs and cats and coarse-grained classes like *red wine* and *traffic light*. Example pictures from this dataset are shown below 2.9. As transfer learning based on this dataset became more popular and successful, a question arose: Which features of this dataset make it so suitable for this task?

Figure 2.9: Sample images from the ImageNet dataset, with examples of similar (fine-grained) classes like two birds of different breeds and coarse-grained classes, like a lamp and a bird. Source: url

A study conducted in [8] attempts to answer this question. The first hypothesis is that the volume of the dataset is relevant to train accurate, general classifiers. The authors compared models that were pretrained on the original dataset and those based on sampled subsets (reduced 2, 4 8 and 20 times). The results show that the more training examples, the better results. The accuracy of the initial classifier occurred to be more dependent on the size of the dataset than the accuracy of classifiers fine-tuned from the base classifier. It is natural that the base classifier's accuracy will depend on the dataset size. Still, the classifier fine-tuned from the base classifier seems to cope with the reduced dataset, and the impact on the accuracy is less detrimental. This is visible in figure 2.10.

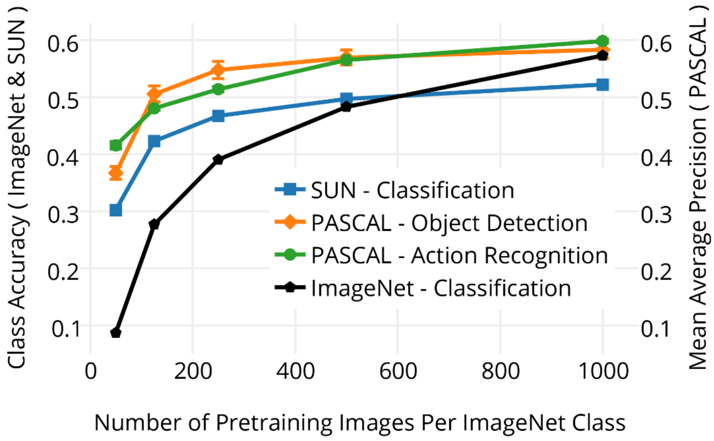

Figure 2.10: Accuracy of the base classifier (black) and classifiers fine-tuned from the base classifier. Image source: [8]

Next experiments examine the label space. The authors study if the granularity of the label space is essential for the problem. To compare the results, the label space is clustered, and 127 classes are derived from the initial 1000 classes. Pre-training with the reduced label space has a minimal negative impact on the accuracy of classifiers fine-tuned from this classifier. This suggests that such a fine division may not be needed.

Finally, the last question is if we train the classifier on the reduced label space with 127 classes, will it be able to distinguish between the fine-grained classes? To examine that, the authors extracted features from the first layers of the networks trained on reduced label space. Then, the authors performed classification with 1-NN and 5-NN models on the extracted feature space but with 1000 classes. The findings are that the k-NN classifier performs 15% worse on a reduced dataset versus the normal dataset.

While the article [8] does not conclude which single feature of the ImageNet dataset makes it so efficient as a source dataset for transfer learning, it is clear that all properties of this dataset are essential for the accuracy of the classifier. We will try to recreate the properties of the ImageNet dataset when creating the source dataset from time series.

2.4.3. Negative transfer in naive transfer learning approach

Negative transfer is a problem in the transfer learning process when the target classifies achieve worse accuracy than without using transfer learning. Transfer learning for time series has been investigated in [6]. The paper's authors tested transfer learning on all 85 datasets from the UCR archive. In the first approach, called *naive transfer learning*, the authors used all pairs of datasets. One dataset from the pair was used as a source dataset to prepare the source classifier for transfer learning. The second dataset was used to fine-tune the former classifier. The architecture used in this approach was the Fully Connected Network, described in section 2.1.3. All layers except for the last one were transferred from the source classifier to the target classifier. The authors compared the accuracy of a fine-tuned classifier to that of a classifier trained without transfer learning.

With respect to different source datasets, target classifiers' accuracies exhibited a high variance. There were cases where the fine-tuned classifiers suffered from transfer learning. On the other hand, for each target dataset, there was always a source dataset beneficial for the classification results. It is shown in the picture 2.11, where for each target dataset, the maximum, median, and minimum accuracies for the different experiments were plotted.

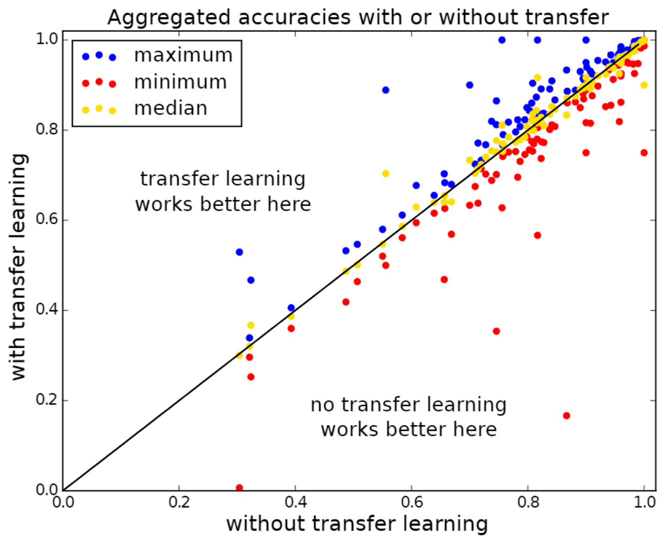

Figure 2.11: Image source: [6]

A negative transfer learning effect can be minimized or avoided by optimally choosing the source dataset/datasets. To improve the transfer learning process, the authors proposed a method of choosing the source dataset based on the DTW distance. This method will be described in 2.4.4.

2.4.4. Inter-dataset similarity based on Dynamic Time Warping Barycenter Averaging

Choosing the source dataset for transfer learning randomly, or by trial and error, can lead to the *Negative transfer* problem, described in the previous chapter (2.4.3). In [6], the authors proposed an inter-dataset similarity measure, that is used to optimally choose the source dataset for transfer learning. The first step of the method is to compute a *prototype* time series per each class, for every dataset that is considered. This reduces the volume of the data, that will be used is the next step. The next step is to compute the distances between each pair of datasets, choosing the minimum DTW distance between prototypes of each class in both datasets. We proceed to describe the algorithm in detail below.

The algorithm uses DTW barycenter averaging [10]. The result of this method is a times series that minimizes the sum of DTW distances to each time series from the dataset.

$$DTW\_average = \arg \min_{X \in 2^{\mathbb{R}}} \sum_{\bar{X} \in \mathcal{X}} DTW(X, \bar{X})$$

The resulting time series does not need to belong to the original dataset $\mathcal{X}$.

The first step of the algorithm is the data reduction step. For each class in the dataset $\{(X_i, y_i) : i = 1, \ldots, N\}$, $y_i = c \in \{1, \ldots, C\}$ the prototype time series P_c is defined as the result of DTW barycenter averaging performed on the subset of $\mathcal{X}$ corresponding to class c. As a result, the original dataset is reduced to C observations.

The next step is to compute the distance between the target dataset and all possible source datasets. The distance between two datasets is the minimum DTW distance between each pair of prototypes from the reduced datasets. The whole algorithm is described in detail in [6].

In [6] the distance was computed pairwise for all available datasets (using train splits). The authors compare the transfer learning performance with a randomly chosen source dataset versus choosing the source dataset that minimizes the former distance to the target dataset. The results are shown below (Figure 2.12).

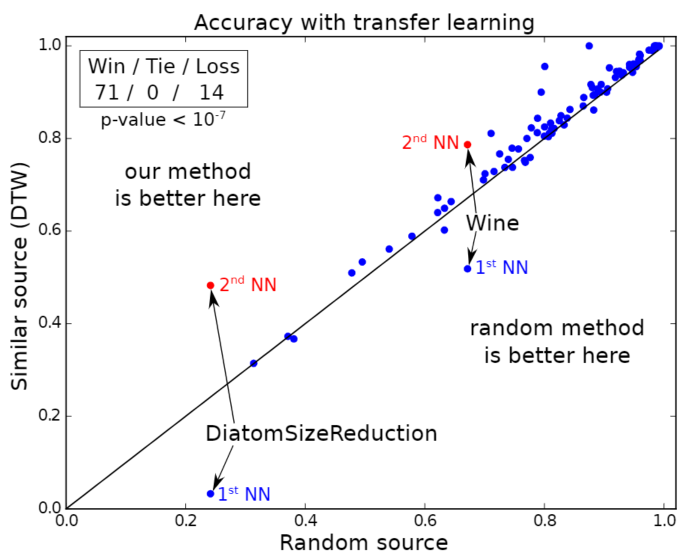

Figure 2.12: Image source: [6]

The result shows that choosing the source dataset for transfer learning based on similarity to the target dataset may lead to benefits. In this work we will try to construct the dataset artificially, by combining, preprocessing, and augmenting the datasets from the UCR archive, to mimic the properties of the ImageNet dataset. The artificial dataset will be composed of multiple datasets. Therefore we rather won't use this similarity to choose one particular dataset. We will take the inter-dataset distance into account when adjusting the distribution of classes in the artificial dataset, to favor classes similar to the target dataset.

Bibliography

[1] Anthony Bagnall, Aaron Bostrom, James Large, and Jason Lines. *The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version.* arXiv, 2016.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[3] Zhicheng Cui, Wenlin Chen, and Yixin Chen. Multi-scale convolutional neural networks for time series classification, 2016.

[4] Hoang Anh Dau, Anthony J. Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn J. Keogh. The UCR time series archive. *CoRR*, abs/1810.07758, 2018.

[5] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML. The ucr time series classification archive, October 2018. `https://www.cs.ucr.edu/~eamonn/time_series_data_2018/`.

[6] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Transfer learning for time series classification. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, dec 2018.

[7] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, mar 2019.

[8] Minyoung Huh, Pulkit Agrawal, and Alexei A. Efros. *What makes ImageNet good for transfer learning?* arXiv, 2016.

[9] Arthur Le Guennec, Simon Malinowski, and Romain Tavenard. Data Augmentation for Time Series Classification using Convolutional Neural Networks. In *ECML/PKDD Work-*

shop on Advanced Analytics and Learning on Temporal Data, Riva Del Garda, Italy, September 2016.

[10] François Petitjean, Alain Ketterlin, and Pierre Gançarski. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition*, 44(3):678–693, 2011.

[11] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. Towards a universal neural network encoder for time series, 2018.

[12] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning, 2018.

[13] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline, 2016.

[14] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. *A survey of transfer learning.* Journal of Big Data, 2016.

[15] Jinsung Yoon, Daniel Jarrett, and Mihaela van der Schaar. Time-series generative adversarial networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[16] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. *A Comprehensive Survey on Transfer Learning.* arXiv, 2019.