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

....

supervisor's signature

....

author's signature

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.

This thesis focuses on proposing, testing, and evaluating multi-source transfer learning for time series classification. We introduce two methods, one of them inspired by the success of ensemble methods in time series classification. Both of those methods apply to most neural network architectures. Both methods were compared to the same network architectures but without transfer learning and proved to improve accuracy and mitigate overfitting.

While transfer learning usually leads to improvements, sometimes *negative transfer* may occur, meaning that the usage of transfer learning is detrimental to the resulting classifier. Negative transfer is especially common when the source domain differs from the target domain or the dataset sizes are small. Given that the datasets present in the time series classification field are usually small, negative transfer may be a threat. This can be mitigated by choosing the source datasets according to the target dataset characteristics.

As the methods proposed in this thesis do not specify which source datasets for the transfer learning are suitable for a given target dataset, we also studied the dataset selection method. We compare selecting the source datasets based on a DTW Barycenter Averaging similarity measure to selecting datasets randomly. Datasets chosen in order to minimize the similarity measure led to improving the target classifier results.

Finally, we studied how many source datasets should be used in the transfer learning procedure. We conducted experiments with 3, 5, and 8 source datasets. All those settings result in satisfactory results, and using 5 source datasets outperformed the two other settings.

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

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 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

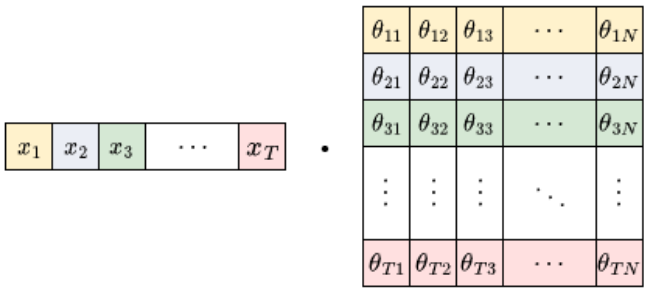

Figure 2.1: The multi layer perceptron architecture treats each entry in the input time series separately.

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:

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

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

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

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.

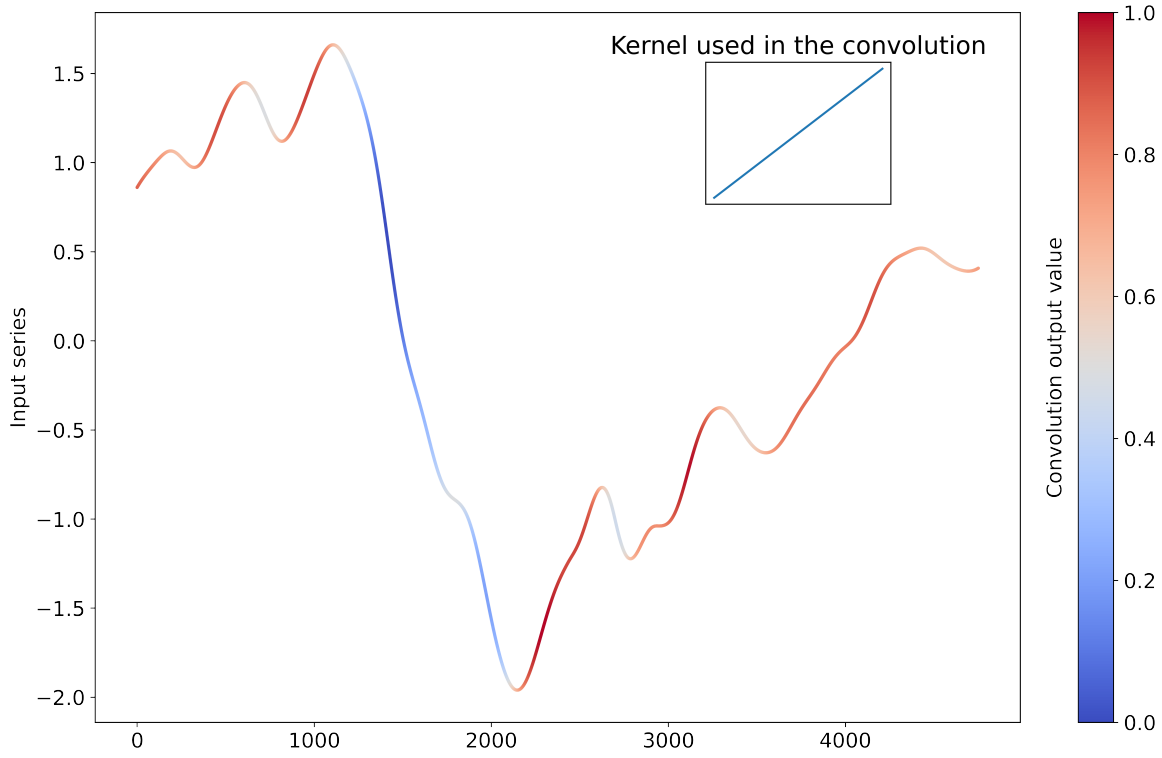

Figure 2.2: Result of applying a "slope" detecting filter to time series. Red regions indicate a higher value in the time series returned by the output convolution.

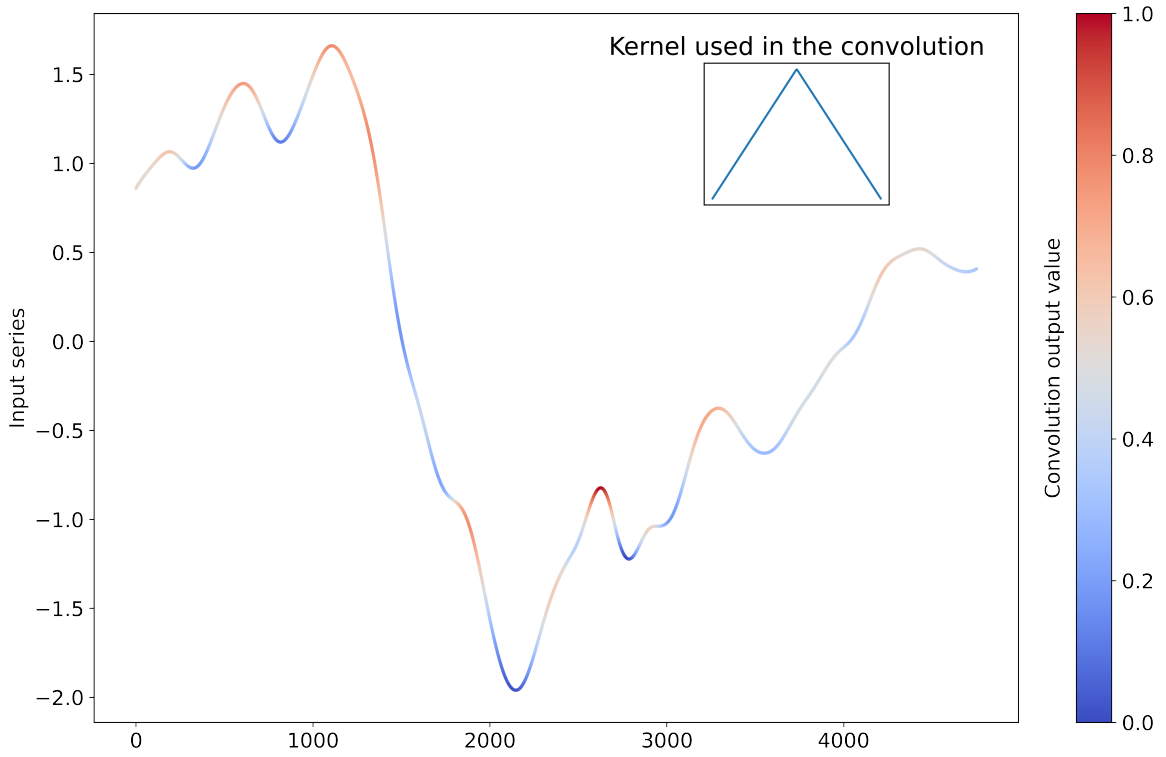

Figure 2.3: Result of applying a "peak" detecting filter to a time series. Red regions indicate a higher value in the time series returned by the output convolution.

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.

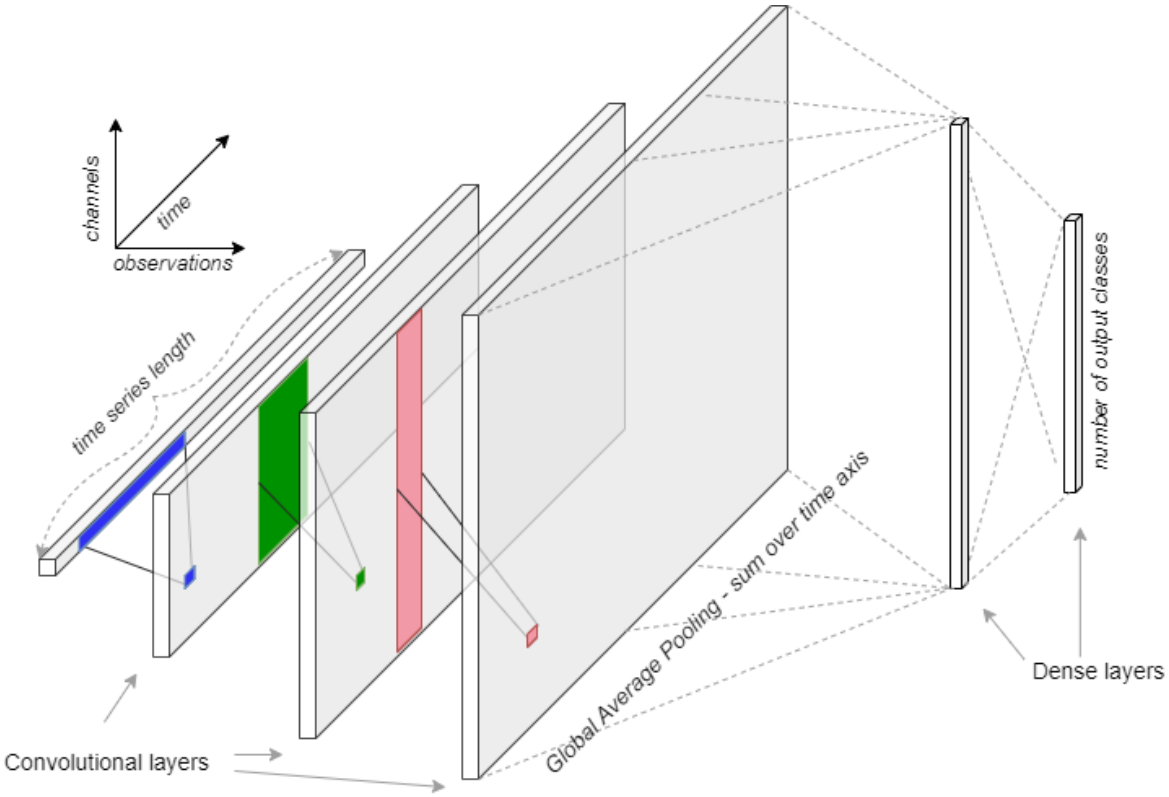

Figure 2.4: Architecture of a Fully Convolutional Network. Source: [7]

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 by Zhiguang, Weizhong et al. in [13]. The Residual Network is suitable for relatively deep architectures 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).

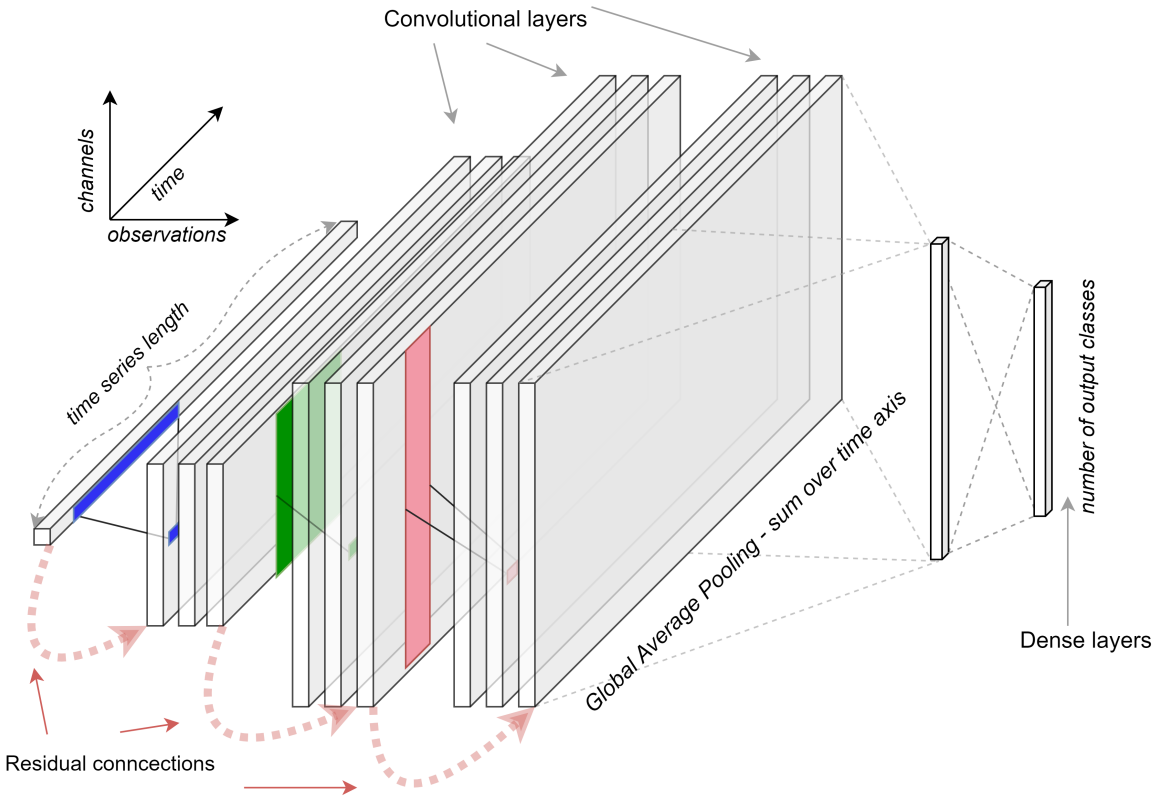

Figure 2.5: Architecture of a Residual Network. Source: [7]

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).

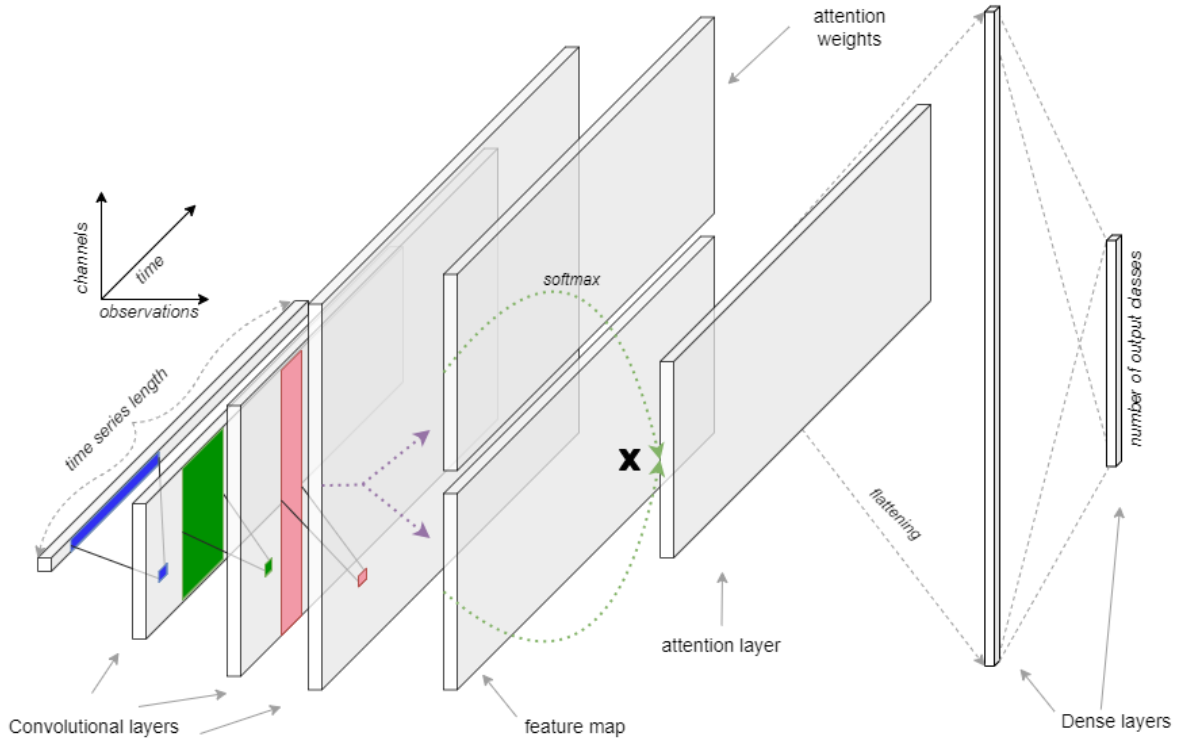

Figure 2.6: Architecture of an Encoder Network. Source: [7]

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 the moment of writing this thesis, it consists of 140 univariate time series datasets. The time series describes various phenomena, like readings from a device or sensor, motion recordings, or food spectrographs. The time series types are summarized below (Table 2.1). We also show several time series samples from different datasets

below (Figure 2.7).

Dataset type (source)	Number of datasets
Audio	10
Device	10
ECG (Electrocardiogram)	7
EOG (Electrooculography)	2
EPG (Electrical penetration graph)	2
Hemodynamics	3
Image	33
Motion	28
Sensor	22
Simulated	9
Spectro	12
Traffic	2

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

All datasets together contain 1122 unique labels. Most datasets differentiate between two labels and have several dozen rows per class. The series length varies from 8 to almost 300 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 other test/train splits and preprocessing the time series accordingly to the method used. We will now describe the datasets in detail, considering all available categories.

- **Audio** - Sound amplitudes of heartbeats and spectrograms of sound produced by insects, cats, dogs, or whales. Classes represent normal and abnormal patients or different species.

- **Device** - This category includes time series describing the power consumption of house devices or summarized population consumption for specific periods. This category is characterized by the combinations of spikes and constant areas in the time series. The data is divided into classes based on different types of devices or different time periods of data collection.

- **ECG** - The data is composed of electrocardiogram readings. The classes correspond to different persons being examined, different diseases detected, the location of sensors, or the time when the data was collected.

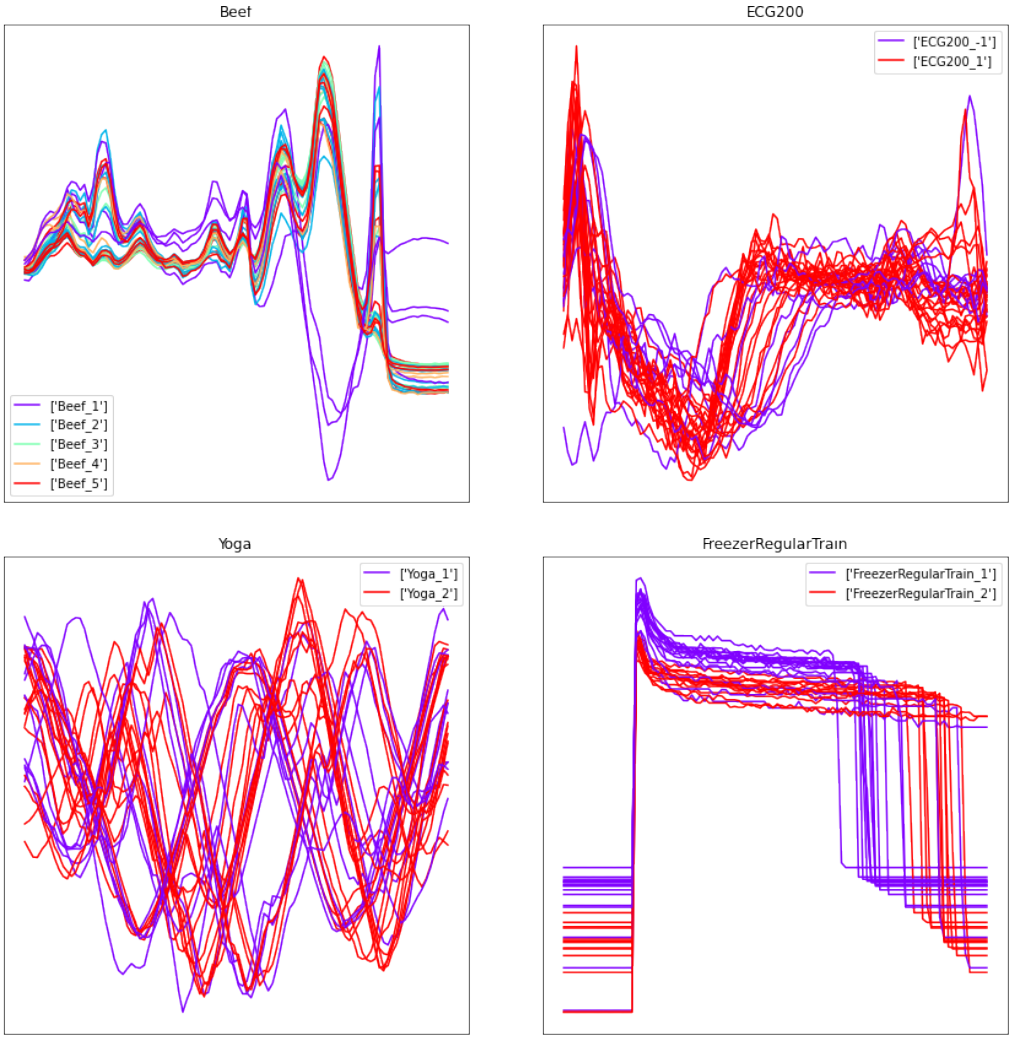

Figure 2.7: Example time series from the UCR archive. The images belong to the following categories: Food, ECG, Image, Device.

- **EOG** - There are two datasets describing electrooculograph readings, measuring the corneo-retinal standing potential between the front and the back of the human eye. The classes distinguish how the reading varied when writing and looking at different characters.
- **EPG** - The electrical penetration graphs measure voltage changes in the electrical circuit that connects insects and their food source.
- **Hemodynamics** - This category consists of time series representing airway, arterial blood, central venous, and pressure measurements. Each class represents one of the pigs that were examined.
- **Image** - The majority of datasets in this category are outlines of images of different shapes. The images represent algaes, arrows, birds, bones, faces, vegetables, written words, fish,

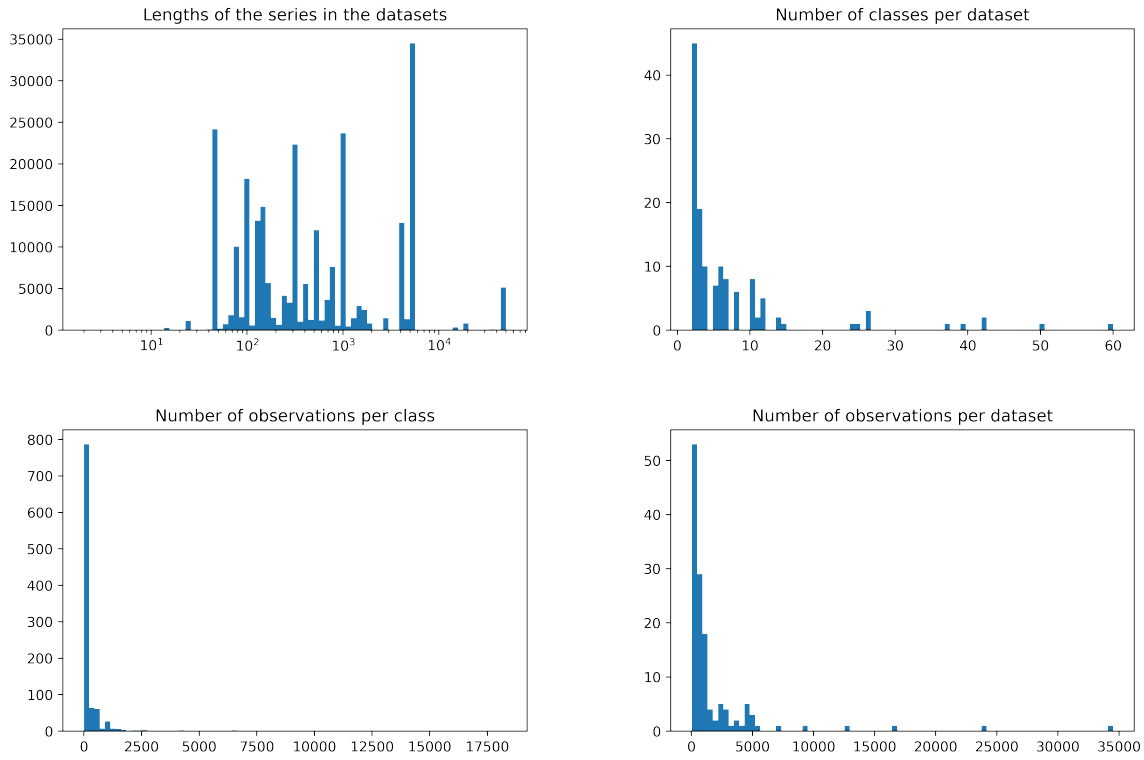

Figure 2.8: Summary of the UCR archive. The first plot is a histogram of time series length. The second histogram shows the number of classes in datasets. The two last histograms summarize the number of rows per class and per dataset.

hands, and leaves. Another type of time series in this category are changes in the pixel intensity or growth of an area on a picture over time.

- **Motion** - Positions of certain points (usually on the human body, but also on a car or worm) over an axis, e.g., while walking, speaking, driving, playing cricket, skating, writing, or performing gestures. One of the datasets uses electromyographic signals instead of point positions. Another dataset is different from others, as it describes the number of pedestrians at different hours.

- **Sensor** - The data includes sensor readings, e.g., earthquake, humidity, temperature sensors, or robots. An example task is to predict an earthquake or to classify the surface the robot was walking on.

- **Simulated** - The time series used in this category were simulated or obtained from an algorithm.

- **Spectro** - This category consists mainly of food spectrograms but also one data set of time series representing surface electromyography which captures the electric activity of groups

of muscle at rest or during activity

- **Traffic** - In this category are two datasets recording pedestrian flow. The classes correspond to weekdays or places.

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

Window Warping was another preprocessing and augmentation method proposed together with the former method in the Time Le-Net model. 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.

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.

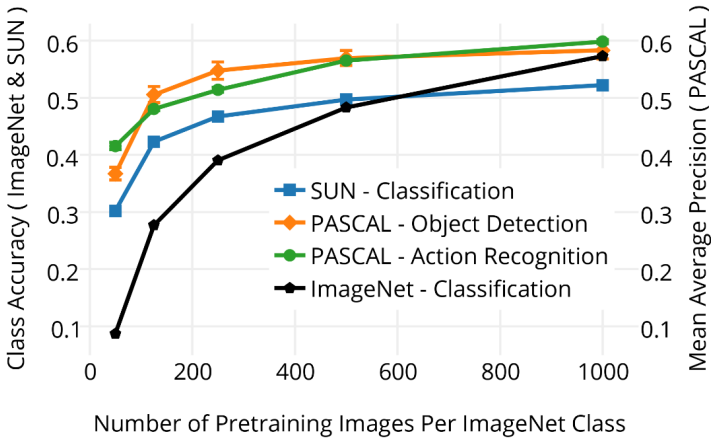

Figure 2.10: Accuracy of the base classifier (black) and classifiers fine-tuned from the base classifier. Image source: [8]

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 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 Convolutional 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.

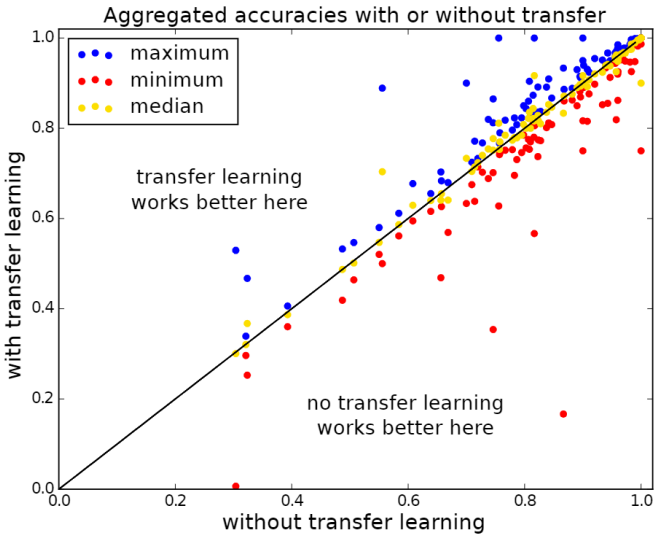

Figure 2.11: Image source: [6]

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 measure 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 choose the source dataset for transfer learning optimally. The first step of the method is to compute a *prototype* time series per each class for every considered dataset. This step reduces the volume of the data that will be used in

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).

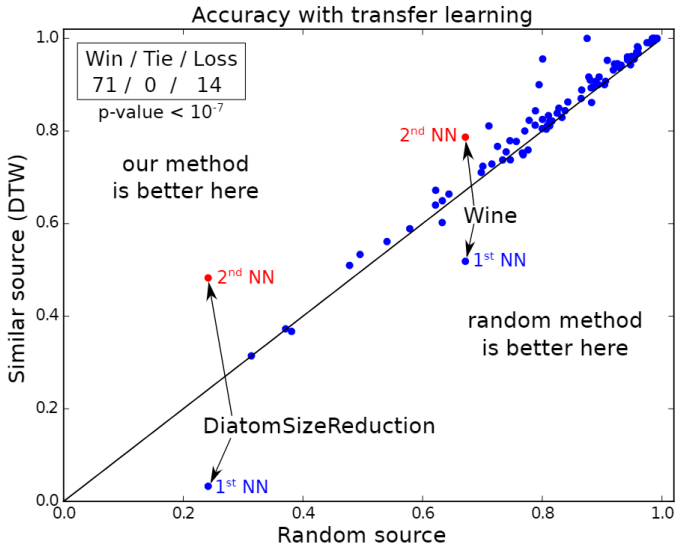

Figure 2.12: Image source: [6]

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 favour classes similar to the target dataset.

3. Metodology

In this chapter, we will cover the technical implementation of the solution. We will introduce methods that aim at inductive transfer learning for time series classification. Most importantly, we will introduce two approaches to multi-source transfer learning - the *baseline* and the *ensemble* approach. We will also cover a selection method for selecting the source datasets given a target dataset.

3.1. Data reading

The data downloaded from the UCR archive is available in two formats, `ts` (as text files) and `arff/weka` (as binaries). In order to speed up the reading and conversion time, all datasets were read, concatenated, and saved as one array of time series (the original lengths were preserved). At this point, only basic preprocessing was done. It included the following steps:

- replacing `NaN` values with zeros
- removing zeros at the end of the time series. This step was done because some of the time series published in the UCR archive were already padded with zeros to have the same lengths. This assumption is not necessary when using a Fully Convolutional Network.
- trimming the series if the time series length exceeded 50000 time steps (applies only for one dataset - `DuckGeese`).

All time series were concatenated to one array of series and saved in a `.npy` format.

3.2. Preprocessing and preparing the data

In order to avoid focusing on one particular form of preprocessing and preparing the data, the preprocessing was done on the fly when training the data. This way, fast experimenting with hyperparameters and methods was possible. In order to achieve it, Keras DataGenerators

were used. DataGenerators allow for defining a Python generator that provides batches of data to the fit method instead of providing a fixed dataframe. Using DataGenerator was useful when dealing with datasets consisting of time series of variable lengths and Fully Convolutional Networks. In such cases, the time series lengths could differ for each batch. For the Fully Convolutional Networks, we implemented `VariableLengthDataGenerator`. The preprocessing steps are described below:

- selecting time series to the batch. The number of series in a batch was equal to 32. The series were chosen so that the classes were uniformly distributed in the batch by resampling.
- normalizing the time series values to standard normal distribution. The normalization is done per one time series
- obtaining time series of the same length within the batch. The length of the longest series in the batch is computed. The target length is equal to the length of the longest series or 3000 if it exceeds. If a time series in the batch is shorter than the target length, then the series is padded with zeros.
- augmentation - 30% of the series in a batch are flipped over the x or y axis with 50% chance.

3.3. Neural network architectures

The experiments were conducted on the Fully Convolutional Network. The network's detailed architecture summary, including layer sizes, is shown below.

```
Model: "FCN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(32, None, 1)]           0

 conv1d (Conv1D)             (32, None, 128)           1536

 batch_normalization (BatchN  (32, None, 128)        512
 ormalization)
```

```
activation (Activation)     (32, None, 128)         0

conv1d_1 (Conv1D)           (32, None, 256)         164096

batch_normalization_1 (Batc  (32, None, 256)        1024
hNormalization)

activation_1 (Activation)   (32, None, 256)         0

conv1d_2 (Conv1D)           (32, None, 128)         98432

batch_normalization_2 (Batc  (32, None, 128)        512
hNormalization)

activation_2 (Activation)   (32, None, 128)         0

global_average_pooling1d (G  (32, 128)              0
lobalAveragePooling1D)

dense (Dense)               (32, 2)                 258

=================================================================
Total params: 266,370
Trainable params: 265,346
Non-trainable params: 1,024
```

The second parameter in output shape is always equal to `None`, because the time series length in a batch is not known in advance.

3.4. Hyperparameters

The hyperparameters used for the training were identical for all experiments. The hyperparameters were usually left as default. We did not experiment with hyperparameter optimization as we focused on the transfer learning approach. The most important hyperparameters are shown in the table 3.1.

Hyperparameter	value
Batch size	32
Augmentation rate (portion of train dataset that was augmented)	0.3
Learning rate	1e-5 with 0.75 decay over 10 000 steps (exponentia
Number of epochs	10

Table 3.1: Summary of hyperparameters.

3.5. Multi-source transfer learning

As single-source transfer learning for time series classification was covered by Fawaz et al. [6], this thesis considers mostly transfer learning with the usage of multiple source datasets. Two approaches of transfer learning were developed, named *baseline approach* and *ensebmle approach*, both of them will be described below. Both of those approaches consider multiple (the number is arbitrary) source datasets, single target dataset, the same neural network architectures, and hyperparameters.

3.6. Baseline approach

The *baseline approach* is a simple extension to the classical transfer learning approach used in deep learning. Usually in deep learning, the neural network is first trained on a large diverse dataset (like ImageNet). The network is usually trained on the source dataset, then the first layers are used in the source classifier.

The idea here is similar, but the source dataset is created artificially from multiple datasets. As most of the datasets in the UCR archive are small, we concatenate them to obtain a bigger dataset. Suppose that we have N datasets $(X_i^{\mathcal{S}}, Y_i^{\mathcal{S}})$, $i = 1, \cdots, N$. The target dataset $(X^{\mathcal{T}}, Y^{\mathcal{T}})$ is created as follows: $X^{\mathcal{T}} = \bigcup_{i=1}^{N} X_i^{\mathcal{S}}$,

$Y = \{prefix_i(y) : y \in Y_i, i = 1, \ldots, N\}$ where $prefix_i(\cdot)$ is a function such that

$$\forall_{i \neq j} \forall_{\bar{y} \in Y_i, \hat{y} \in Y_j} \quad prefix_i(y) \neq prefix_j y$$

In our case, the prefix function given a class name (as a string), simply transforms the string by adding a prefix with the dataset name. This way the classes remain distinct after concatenation. The number of classes in the resulting dataset is equal to the sum of the number of classes in each of the datasets. The examples that correspond to the same dataset should be intuitively similar to each other (analog of the fine-grained classes in ImageNet) while examples from different datasets

cover more latent space (analog of the coarse-grained classes in ImageNet). The concatenation process is shown in figure 3.1

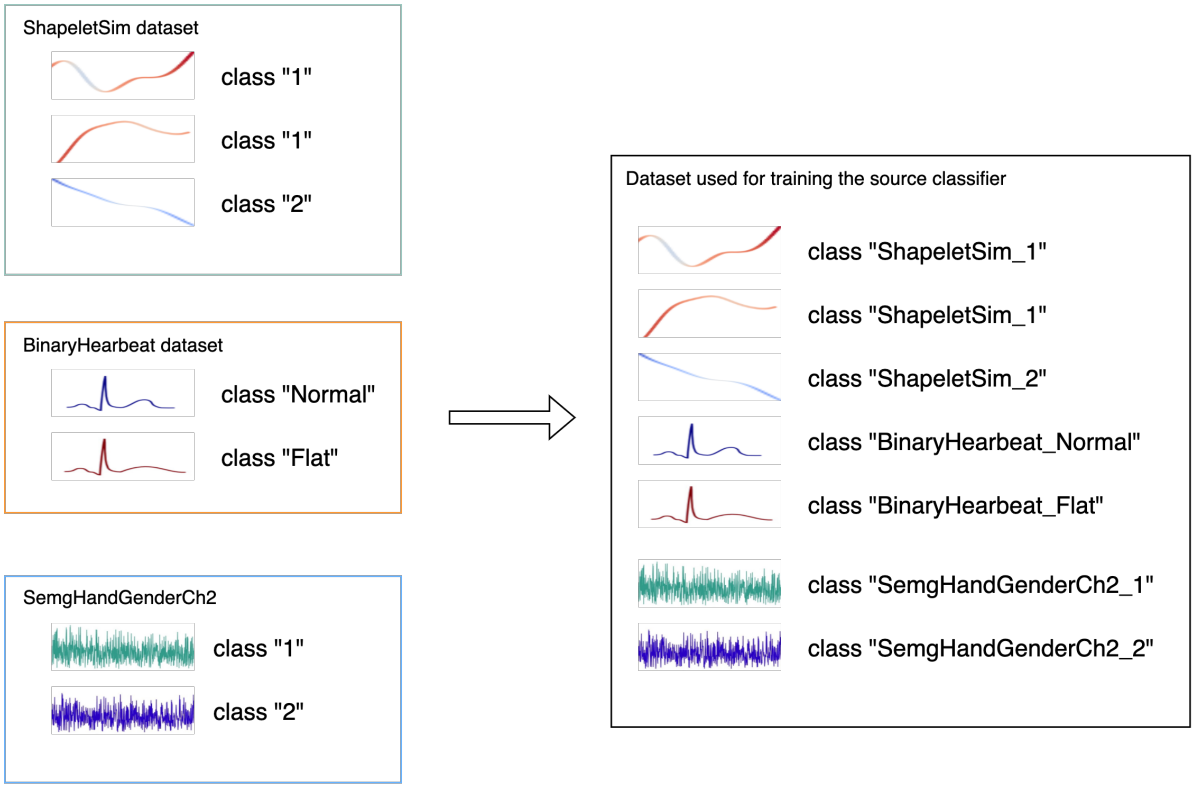

Figure 3.1: Illustration of preparing the data for the source classifier in the *baseline* approach.

The data batches that are fed to the model are processed as described in the previous chapter, including length unification within the batch, normalization, and random augmentation of 30% of the observations.

After training the classifier on the artificial, concatenated dataset, we obtain a neural network model. The network (specifically the weights) is then used as a starting point to train on the target dataset. The target classifier is created by removing the last layer, which outputs the probabilities for each class. The layer is replaced by a new layer, with randomly initialized weights and output size equal to the number of classes in the last layer. The whole network is then retrained on the target dataset. All weights are updated in the training procedure (freezing the weights is not used).

We compare the results by predicting the classes on the test split of the target dataset. The metrics used for comparison will be explained in detail in the Results chapter. We compare the metrics to results obtained by training a network of the same architecture but without transfer learning (with random weights).

3.7. Ensemble approach

The second approach proposed in this thesis is inspired by the success of ensemble models (e.g. HIVE-COTE) in time series classification. Small dataset sizes common in time series classification tend to increase the variance error of the classifiers. Therefore the ensemble method is often proposed to decrease the variance and increase the accuracy of predictions. This intuition is resembled in this approach.

The previous study on transfer learning with a single source dataset [6] showed that though transfer learning generally increases the accuracy, for a lot of experiments the accuracy was decreased. If a dataset differed significantly from the target dataset, a classifier pre-trained on such dataset wouldn't adapt to the target dataset, thus decreasing the accuracy. The same situation is probable with multi-source transfer learning if some of the datasets are not suited for the particular target dataset. In order to mitigate the negative contribution, the ensemble method is used.

Let's assume that we want to perform transfer learning given N source datasets $(X_1^{\mathcal{S}}, y_1^{\mathcal{S}}), \ldots, (X_N^{\mathcal{S}}, y_N^{\mathcal{S}})$ and a target dataset $(X^{\mathcal{T}}, y^{\mathcal{T}})$. The model is created from N source models. Model M_i is a classifier (e.g. a Fully Convolutional Network) trained on the dataset $(X_i^{\mathcal{S}}, y_i^{\mathcal{S}})$. The hyperparameters used for experiments are described in section 3.4 and do not differ from the hyperparameters used for the *baseline approach*.

Let's recall the notion from the section 2.1.1 that a model M_i is a composition of layers,

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

To shorten the notation, let's group the first layers together:

$$E(X; \Theta^i, B^i) = L_{M-1}^i(\ldots L_2(L_1(X; \theta_1^i, \beta_1^i); \theta_2^i, \beta_2^i); \theta_{M-1}^i, \beta_{M-1}^i)$$

thus

$$M_i(X; \theta_1^i, \ldots, \theta_M^i, \beta_1^i, \ldots, \beta_M^i) = L_M^i(E(X; \Theta^i, B^i); \theta_M^i, \beta_M^i)$$

We assume that the architecture of subsequent layers except for the last one is the same for all models M_i, $i = 1, \ldots, N$. Therefore only the last layer has the superscript ^{i}. The last layer output size is equal to the number of classes in dataset i, therefore may be different for every model. We also assume that the first layers architecture is the same for all models. Models agnostic to the input size (like FCNs) are advantageous here, as their architecture does not depend on the input size. As for other models, we assume a fixed size of the input data achieved by preprocessing the input data. The weights θ_k^i and biases β_k^i are obtained from training on

dataset i. The target model M is created as follows:

$$M(X) = \frac{1}{N}\sum_{i=1}^{N} \hat{L}_M(E(X;\Theta^i, B^i); \hat{\theta}^i_M, \hat{\beta}^i_M)$$

The weights $\theta^i_1, \ldots, \theta^i_{M-1}$ and biases $\beta^i_1, \ldots, \beta^i_{M-1}$ are obtained from training the source models. The last layer $\hat{L}_M$ is "new", meaning that the weights are randomly initialized. The new corresponding weights $\hat{\theta}^i_M$ and biases $\hat{\beta}^i_M$ have proper dimensionality (corresponding to the number of classes in the target dataset). This model is then retrained on the target dataset. All weights are updated in the training procedure (freezing the weights is not used).

A graphical example of the process can be found in the figure. 3.2.

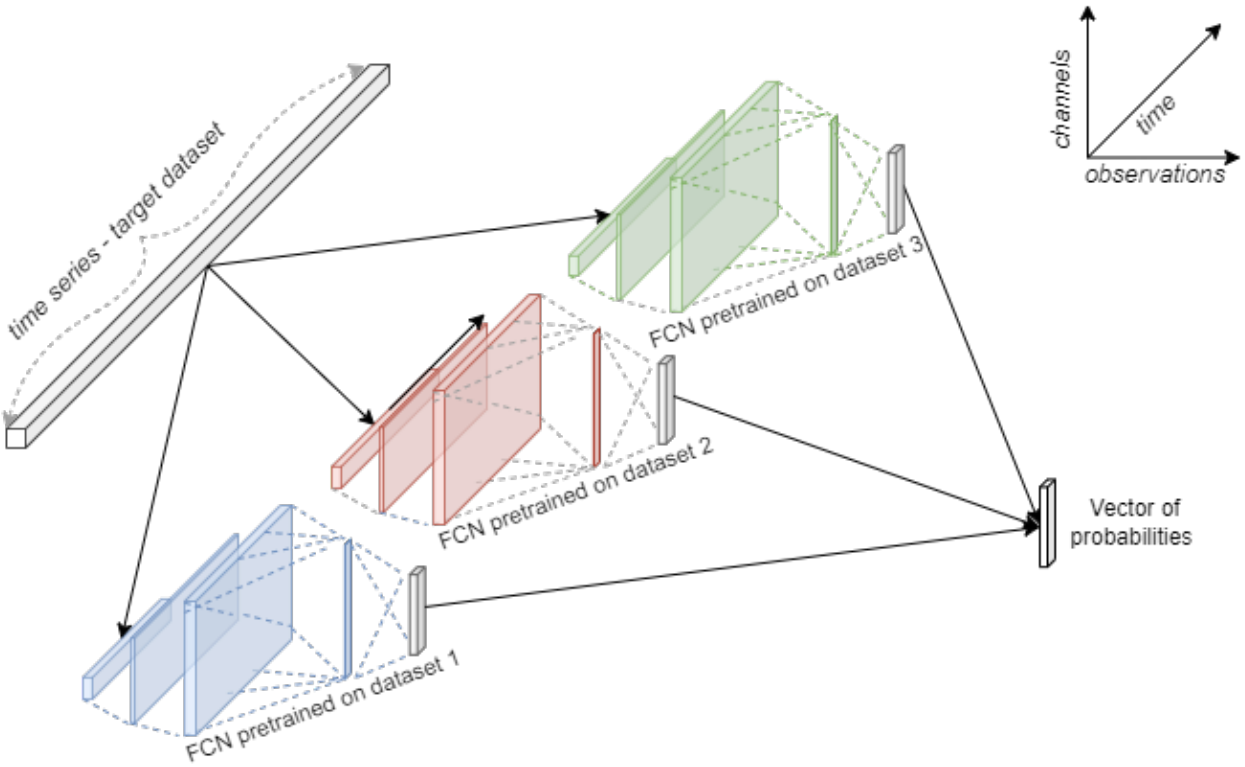

Figure 3.2: Illustration of the *ensemble* approach based on $N = 3$ source models (Fully convolutional networks in this case). If a layer is colored, it means that the weights in this layer are initialized by pre-training on one of the source datasets. The last layer in each model is new.

3.8. Selecting the source datasets for multi-source transfer learning

In both multi-source learning approaches described above (*baseline* and *ensemble* approaches) we don't impose any restrictions on the source datasets themselves. Neither of the two approaches doesn't include any rules on how the source datasets correspond to the target dataset. In the

experiments conducted in this thesis, two approaches for picking the datasets are proposed. In the first approach, we pick randomly N datasets from a given category corresponding to the target dataset. To illustrate, let's say that the objective is to train a target classifier on the `InlineSkate` dataset, which belongs to the `Motion` category in the UCR archive. Then the source datasets are picked randomly from the given category, which could be `UWaveGestureLibraryX, AsphaltPavementType, ToeSegmentation2, CricketZ, UWaveGestureLibraryZ`.

The second approach described was based on a similarity measure. The inter-dataset similarity measure called DTW barycenter averaging was described in 2.4.4. The intuition behind the second approach is that usually transfer learning benefits if the domains are similar. To measure the similarity, the DBA distances were computed for all pairs of datasets in a given category. Then, for a given target dataset, N datasets minimalizing the distance to the target dataset were used for pretraining the source network (either in *baseline* or *ensemble* approach). The dataset still belonged to the same UCR category as the target dataset.

3.9. Environment and code structure

The solution is written in Python 3.8, mainly utilizing the library `Keras` from the `TensorFlow` library. The structure of the code is as follows:

```
./MASTER-THESIS/SRC
+---data
|   ...
+---experiments
|   +---no_transfer_learning
|   +---transfer_learning
+---mlflow_logging
+---models
+---preprocessing
+---reading
+---results
|   +---baseline
|   +---baseline_vs_ensemble
|   +---dba_vs_random
|   +---ensemble
|   +---ensemble_dba_3_vs_5
```

```
|   +---ensemble_dba_5_vs_8
+---selecting
```

The folders contain the following content:

- `data` - `ts` and `arff` files downloaded from the UCR archive, serialized `keras` models to use in the ensemble (see section 3.7) and `numpy` files containing all series from UCR archive concatenated to one dataframe (see section 3.1)
- `experiments` experiments corresponding to scenarios described in 3.7 and section 3.6 and without the use of transfer learning for comparison purposes.
- `mlflow_logging` - utility functions to write metrics and parameters of the experiments and runs to MlFlow
- `models` - model architecture definitions in `keras` syntax
- `preprocessing` - utility functions for preprocessing purposes, definitions of DataGenerators (see 3.2.
- `reading` - script for reading `ts` and `arff` files, creating the concatenated dataset (see 3.1)
- `results` - scripts for preparing results (as plots or sole values) of comparing experiments conducted by scripts in the `experiments` directory. The experiments are further described in 3.7 and section 3.6. The results are described in chapter 4
- `selecting` - source dataset selection methods (see 3.8)

The experiments that consisted of model training were run in Google Colab with GPU. Other scripts, for example, to calculate DBA similarity measures or to summarize results gathered in MlFlow were run locally.

4. Results

The results and metrics of classifiers obtained from training networks with transfer learning were compared to the results obtained without transfer learning. The performance of the target classifier in the *ensemble* approach was compared to an ensemble model with the same amount of component models in the ensemble.

4.1. Experiments

Each experiment recorded in MlFlow corresponds to one training setting. An example setting may be e.g.

- training a network using the *ensemble approach*, with 5 datasets picked using the DBA similarity measure
- training a network using the *baseline approach*, with 5 source datasets picked randomly
- training an ensemble of FCNs without transfer learning

Following the MlFlow convention, each experiment consisted of several runs, each corresponding to one target dataset from the category `MOTION` and `IMAGE`. This is shown on screenshot 4.1. When comparing averaged results, then the result was averaged on all runs (61 runs for each experiment).

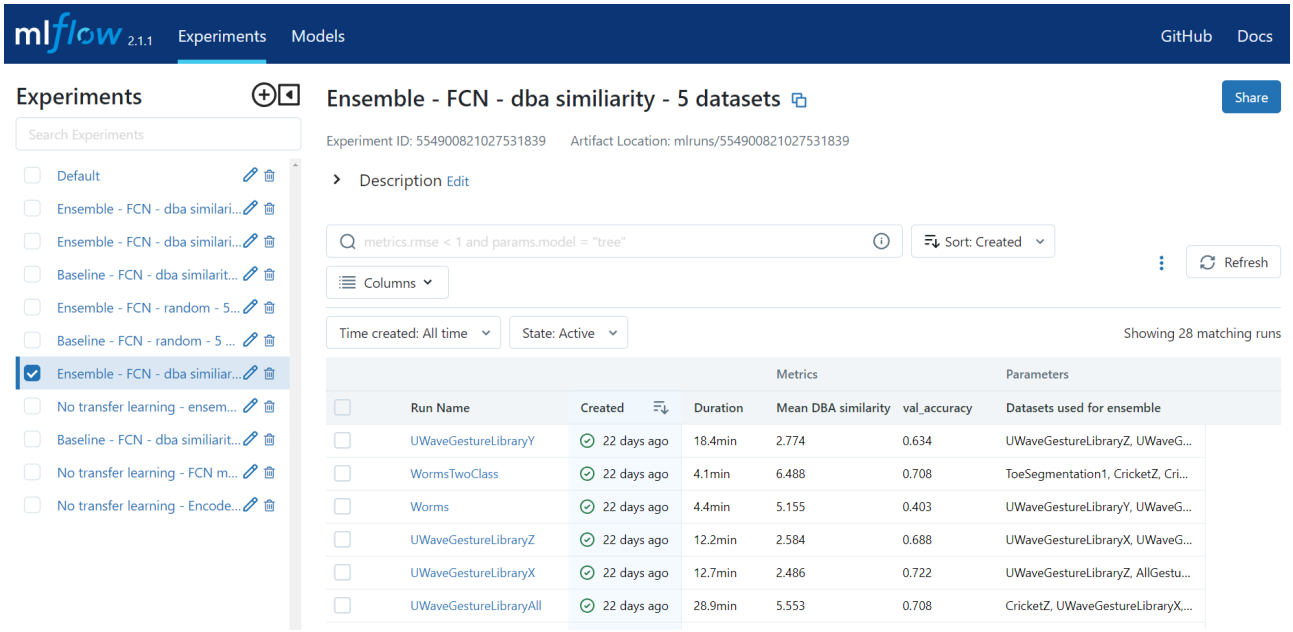

Figure 4.1: The structure of experiments recorded in MlFlow. Each experiment corresponds to a different setting, and each run corresponds to a model trained on a different target dataset (and consequently different source datasets).

4.2. Evaluation of the *baseline* approach

The *baseline* approach, in which a single Fully Convolutional Network was pre-trained and fine-tuned, was compared to results obtained from training a single FCN model on the target dataset only. For this comparison, we used a Fully Convolutional Network as the model and 5 source datasets were selected to minimize the DBA distance to the target dataset. Figure 4.2 shows the accuracy and loss in each epoch, averaged over all experiments conducted.

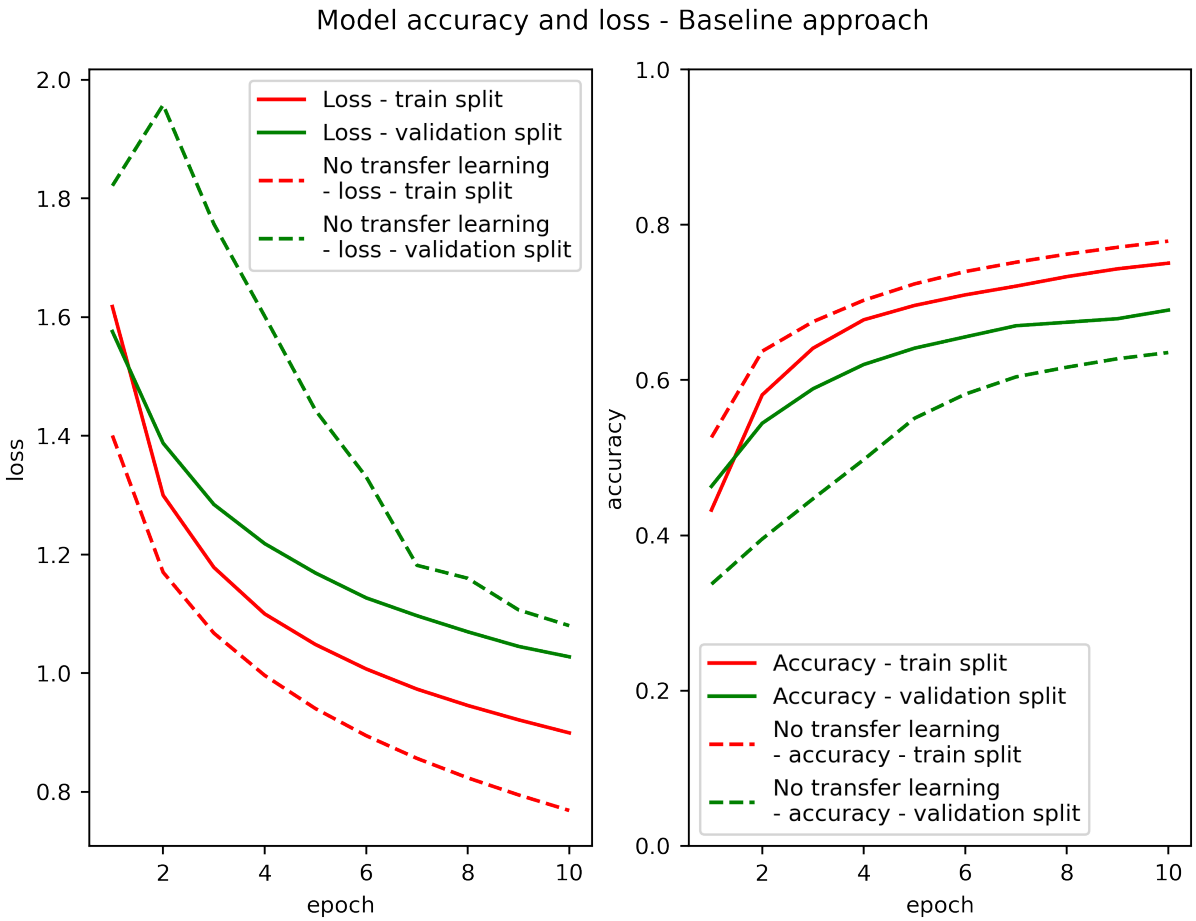

Figure 4.2: Averaged accuracy and loss obtained from transfer learning with the *baseline* approach, compared to a single FCN trained only on the target dataset.

The greatest improvement is visible in the first epochs of the training process, which suggest that the network uses the knowledge learned from the previous task. As the training continues, the advantage is less significant. In figure 4.3 we show a comparison win/tie/loss diagram for the fifth and tenth epochs. We also see that the gap between accuracy/loss on the validation and test split is smaller with transfer learning. This can mean that the model is less prone to overfitting and hence generalizes better, due to exposure to more diverse examples in the pre-training phase.

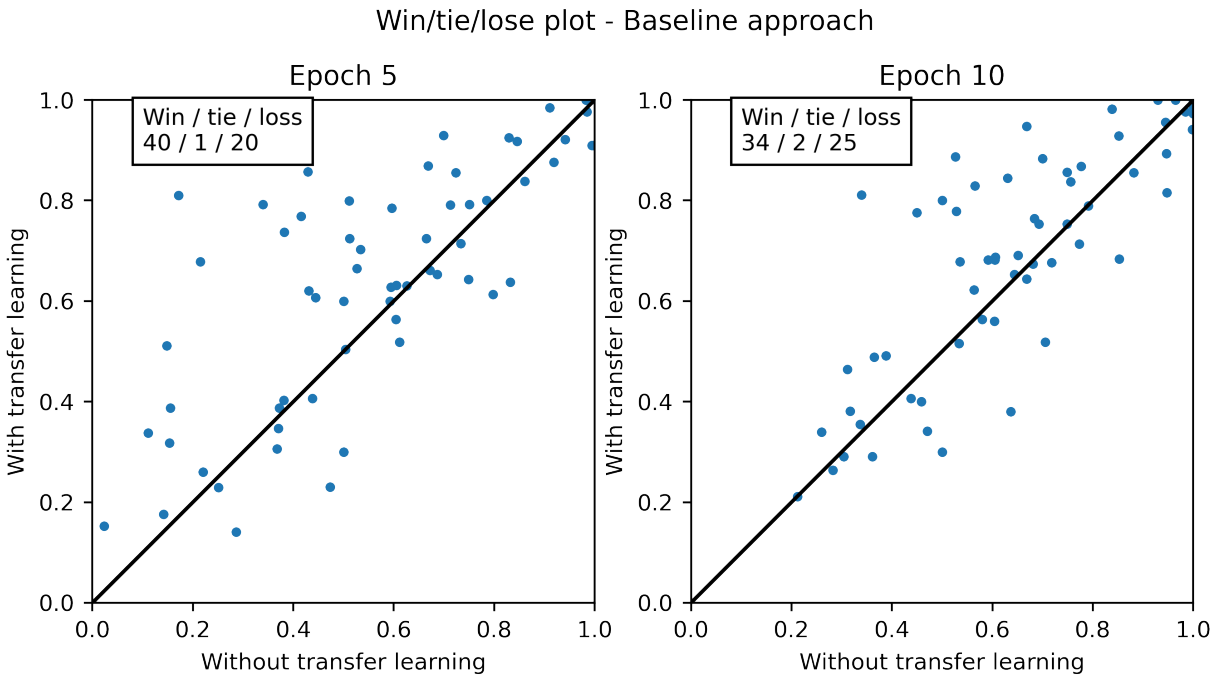

Figure 4.3: Win/tie/loss plot. This plot shows the validation accuracy in the 5th and 10th epoch for each run (corresponding to a target dataset). The y axis corresponds to baseline approach, and x-axis to the approach without transfer learning. If a dot is above the diagonal, then it means that the transfer learning was beneficial.

4.3. Evaluation of the *ensemble* approach

In the *ensemble* approach, we created an ensemble model created from 5 FCN models pre-trained on different datasets. The pre-trained ensemble was fine-tuned on the target dataset and the results were compared to a model whose architecture was identical to the ensemble, but the weights were initialized randomly. Transfer learning led to improvements in accuracy and loss, similarly as in the *baseline* approach. Figure 4.4 shows the comparison of accuracy and loss in each epoch, averaged over all runs (each run corresponds to a target dataset).

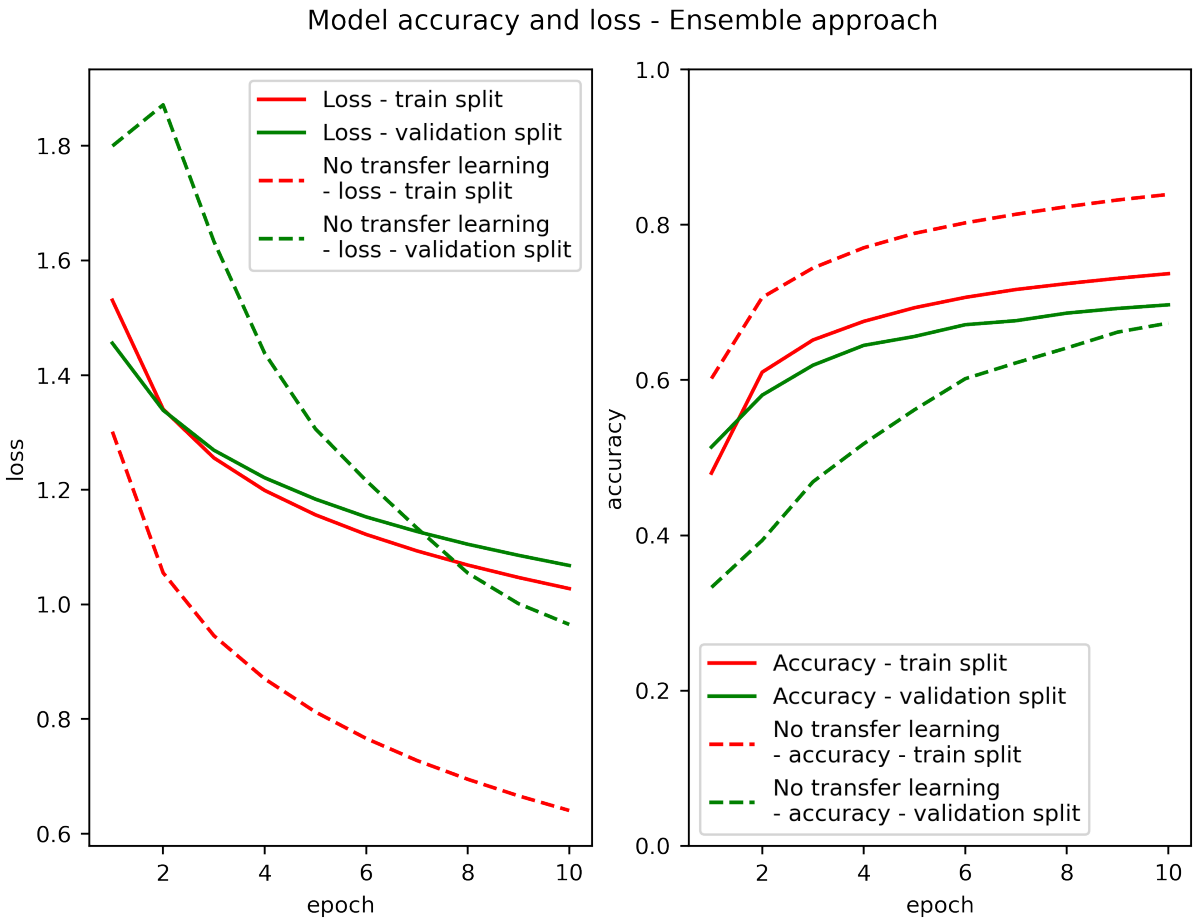

Figure 4.4: Averaged accuracy and loss obtained from transfer learning with the *ensemble* approach, compared to an ensemble of FCN models trained only on the target dataset.

We also present a win tie loss diagram for the *baseline* approach in figure 4.5. We see that in the final epoch, a majority of the datasets didn't benefit from transfer learning, but if they did, the benefit was relatively bigger than in the *baseline* approach.

The model used in the ensemble approach consisted of more parameters, thanks to ensembling. This together with the small sizes of datasets in the UCR archive might lead to overfitting. As we see in the plot, in the model trained without transfer learning, the gap between accuracy and loss for validation and test split is big. This may indicate that the models were overfitted. Exposing the models to a larger amount of data in the pre-training phase reduces overfitting. We see that for the transfer learning results, the gap between metrics for validation and train split is smaller. Also, the accuracy of the train split is smaller, which suggests that the overfitting is mitigated.

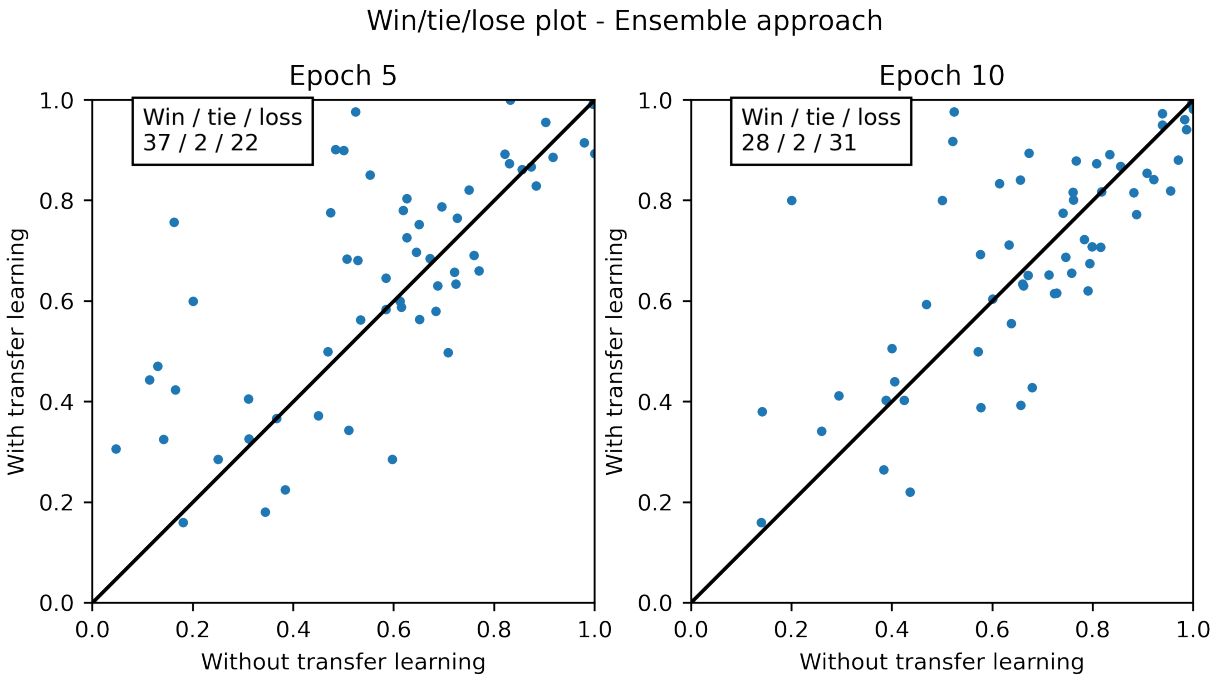

Figure 4.5: Win/tie/loss plot. This plot shows the validation accuracy in the 5th and 10th epoch for each run (corresponding to a target dataset). The y-axis corresponds to the *ensemble* approach and the x-axis to the approach without transfer learning. If a dot is above the diagonal, then it means that the transfer learning was beneficial.

4.4. Comparison of *baseline* and *ensemble* approaches

We see that in both approaches the results were improved, compared to training a network from scratch. Figure 4.6 shows the accuracy and loss per each epoch averaged over several runs (corresponding to target datasets)

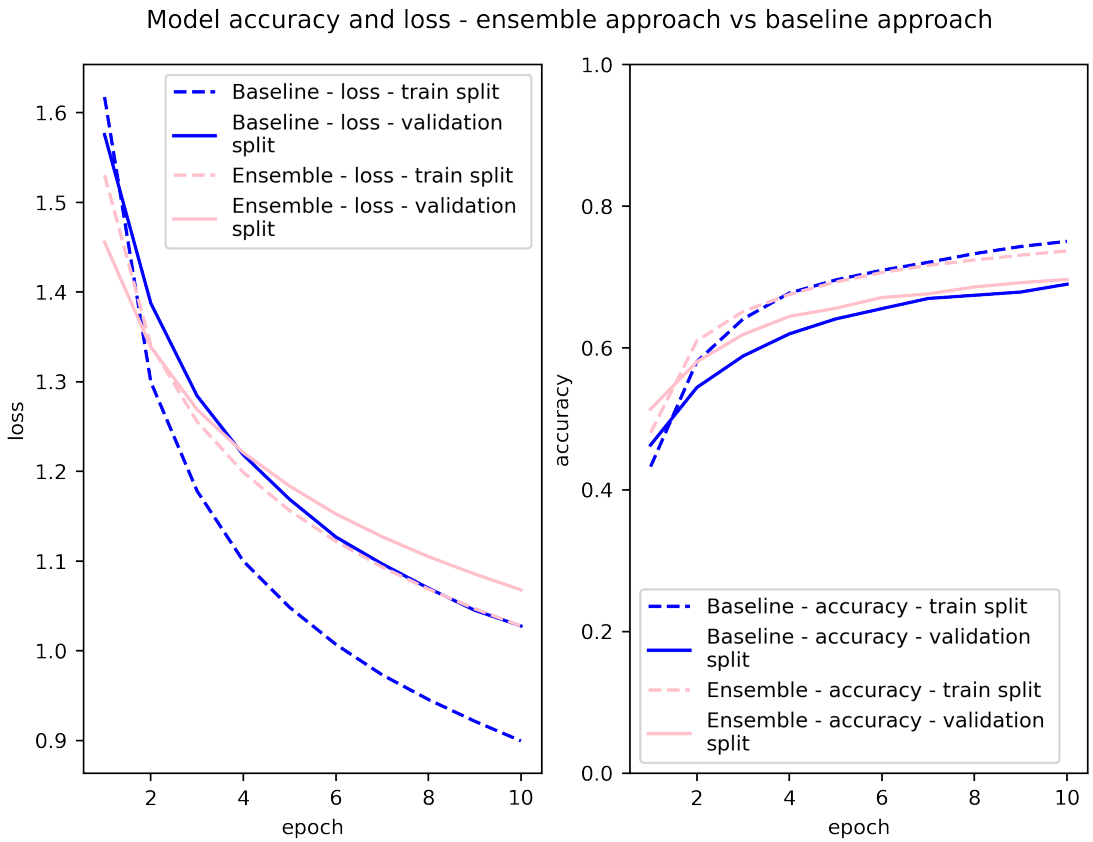

Figure 4.6: Comparison of loss and accuracy for *baseline* approach and *ensemble* approach.

Similarly as in the two previous sections, where we compared *ensemble* and *baseline* approaches to training without transfer learning, figure 4.7 shows a win / tie / loss diagram for 5th and 10th epoch comparing the *baseline* and *ensemble* approaches.

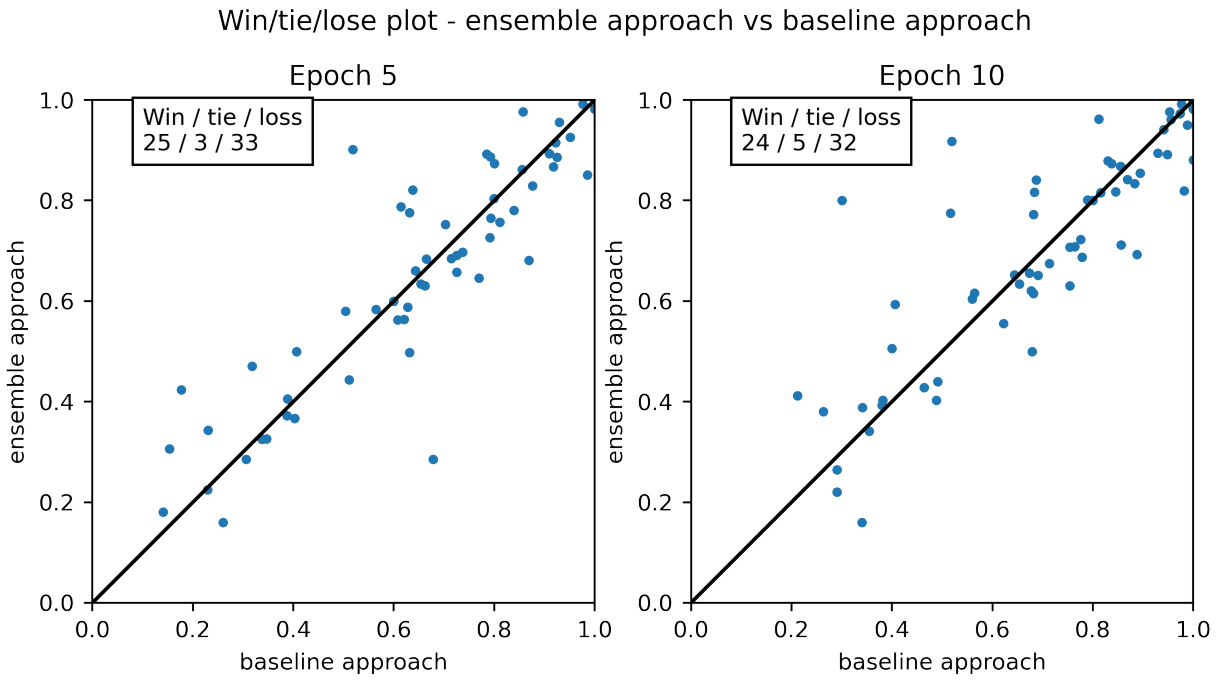

Figure 4.7: Win /tie / loss diagram for *baseline* and *ensemble* approach. Dot above the diagonal line means that the run (corresponding to training on a particular dataset) achieved better results with the *ensemble* approach.

Other than the differences in qualitative results, the computation time and sizes of the classifiers differ in both approaches. For the baseline approach, the model is smaller, and it takes a little bit less time to pre-train a single model with a large amount of data than to pre-train a few small separate models with smaller data. Therefore, the time needed to pre-train the network in the *baseline* approach is longer than in the *ensemble* approach.

Training the model used in the *ensemble* approach is composed of two steps. In the first step each component, which is a single FCN model, is trained on a single dataset, corresponding to each source dataset selected for the target dataset. Hence it is a single dataset and a single model, this could be parallelized, as the models and datasets and independent. The second step consists of training the model on the ensemble created from the pre-trained components on the target dataset. The resulting model is bigger than the final classifier in the *baseline* approach. Summing this up together, pre-training the ensemble components and training the network takes longer than in the *baseline* approach. The training procedure in the baseline approach consists of training the source (single) model on the combined dataset and retraining it on the target dataset. Figure 4.8 shows the comparison of training time for both approaches. The training time shown on the plot is the sum of time needed for the pre-training step and fine-tuning step.

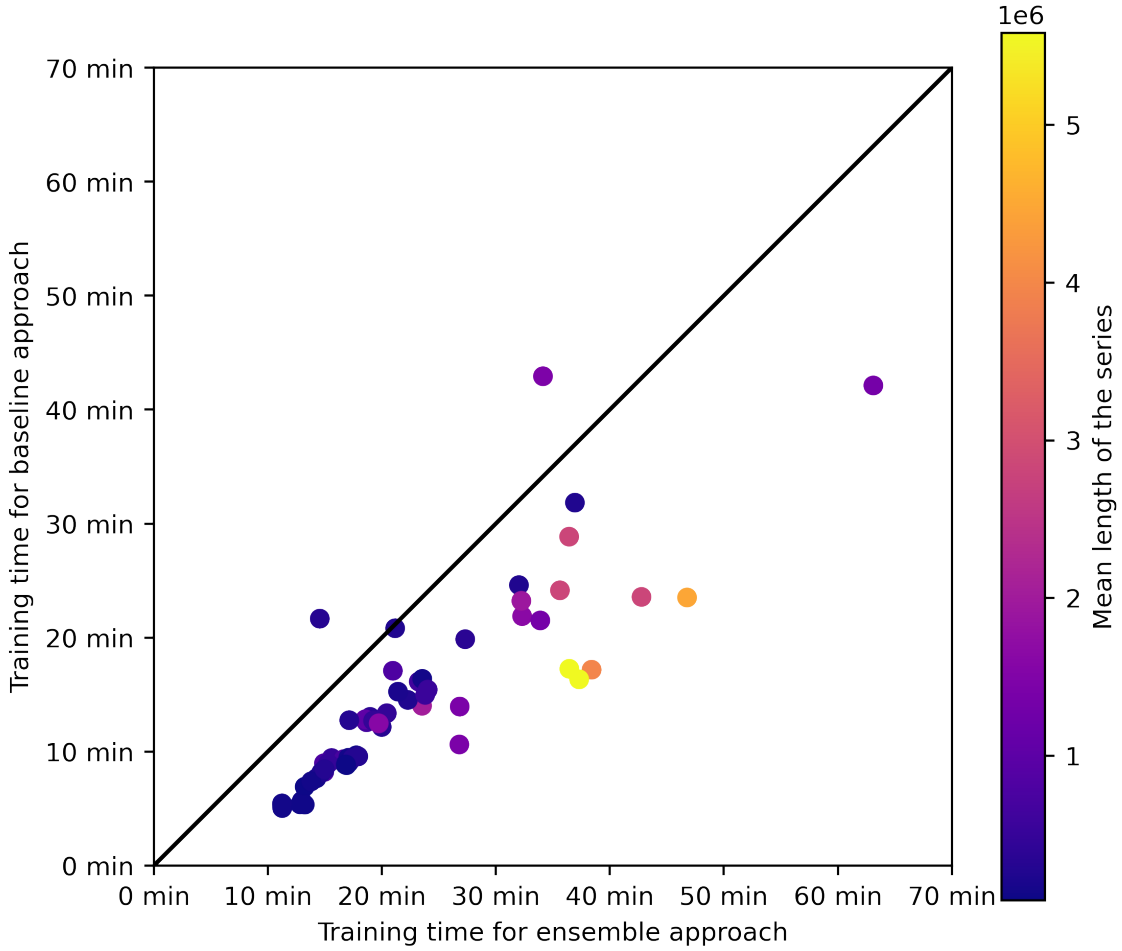

Figure 4.8: The total time needed to train the source classifier/classifiers and the target classifier in *baseline* and *ensemble* methods. The color shows the total length of the series (source and target dataset).

It is worth mentioning, that during the experiments conducted for the sake of this thesis, a lot of time was saved, by saving the components pre-trained for the *ensemble*. If a component was trained once on a given source dataset, it was reused every time an ensemble model was created of it. It saved time during the experiments, but the time saved was not deducted from the time shown on figure 4.8.

When it comes to the size of the model, the model used for *baseline* approach consisted of c.a. (depending on the number of classes) 266000 parameters, while the model used for *ensemble* approach (created from 5 components) consisted typically of 1332000 parameters. The model weights used for *baseline* occupied 3.22 MB after saving on disk (as a TensorFlow Checkpoint),

and the model weights used for the ensemble approach (created from 5 components) used 16.04 MB of disk memory.

4.5. Evaluation of method used for selecting the source datasets for the ensemble

As mentioned in 3.8, the datasets used for multi-source transfer learning were selected in two ways: randomly or based on the DBA similarity measure. This section contains a comparison of those two methods. The selecting method is independent of the model, thus it can be used for both *ensemble* and *baseline* approach. Unfortunately, this evaluation is performed only using the *ensemble* approach, given the time and computational limitations.

Figure 4.9 shows the averaged results using random selection and selection based on the DBA similarity measure.

As we see, selecting the datasets based on the similarity measure led to slight improvement.

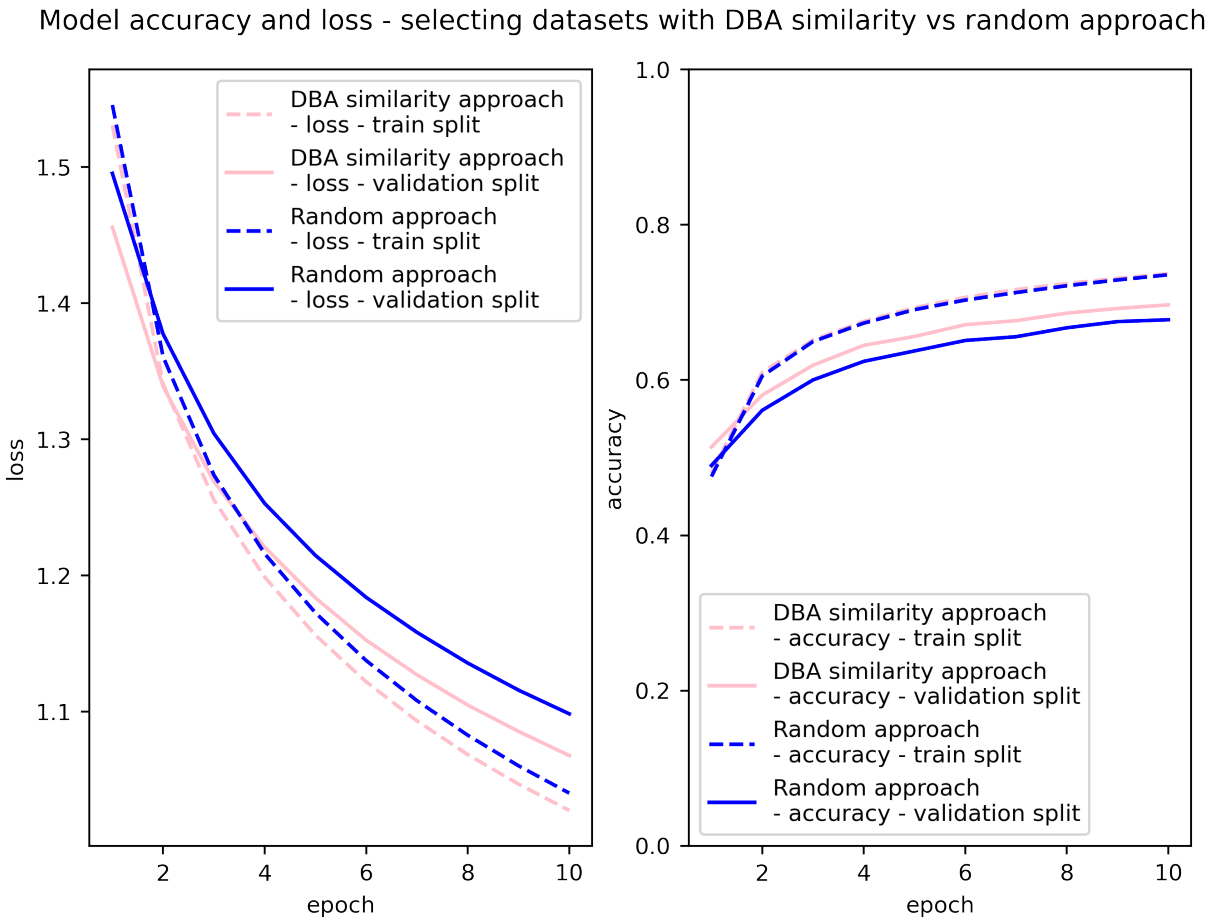

Figure 4.9: Averaged accuracy and loss per each epoch for the ensemble approach, when selecting the source datasets randomly and using the DBA similarity approach.

The win tie loss diagram also indicates an advantage of the DBA selection approach (figure 4.10

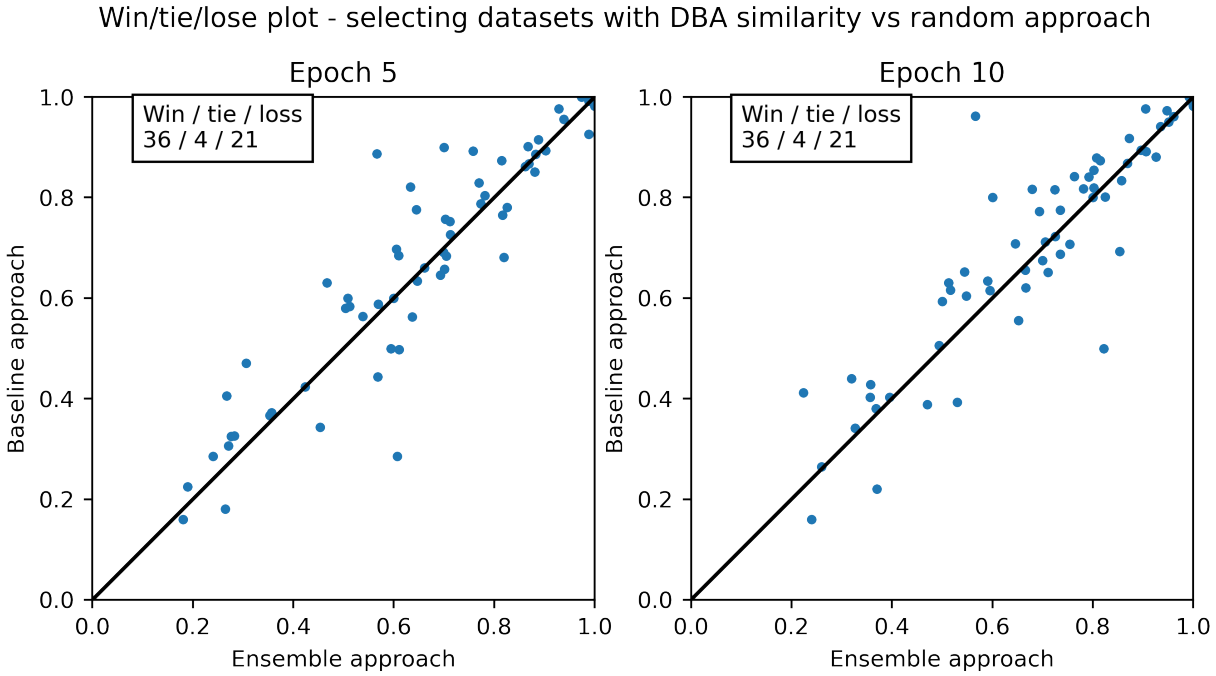

Figure 4.10: Win tie loss diagram comparing selecting the source datasets randomly vs with the DBA similarity approach.

There is still room to improve the similarity measure. In order to understand the improvements in accuracy when using similarity score, the correlation between the target classifier accuracy and the dba similarity has been examined. The scatterplot is shown on plot 4.11.

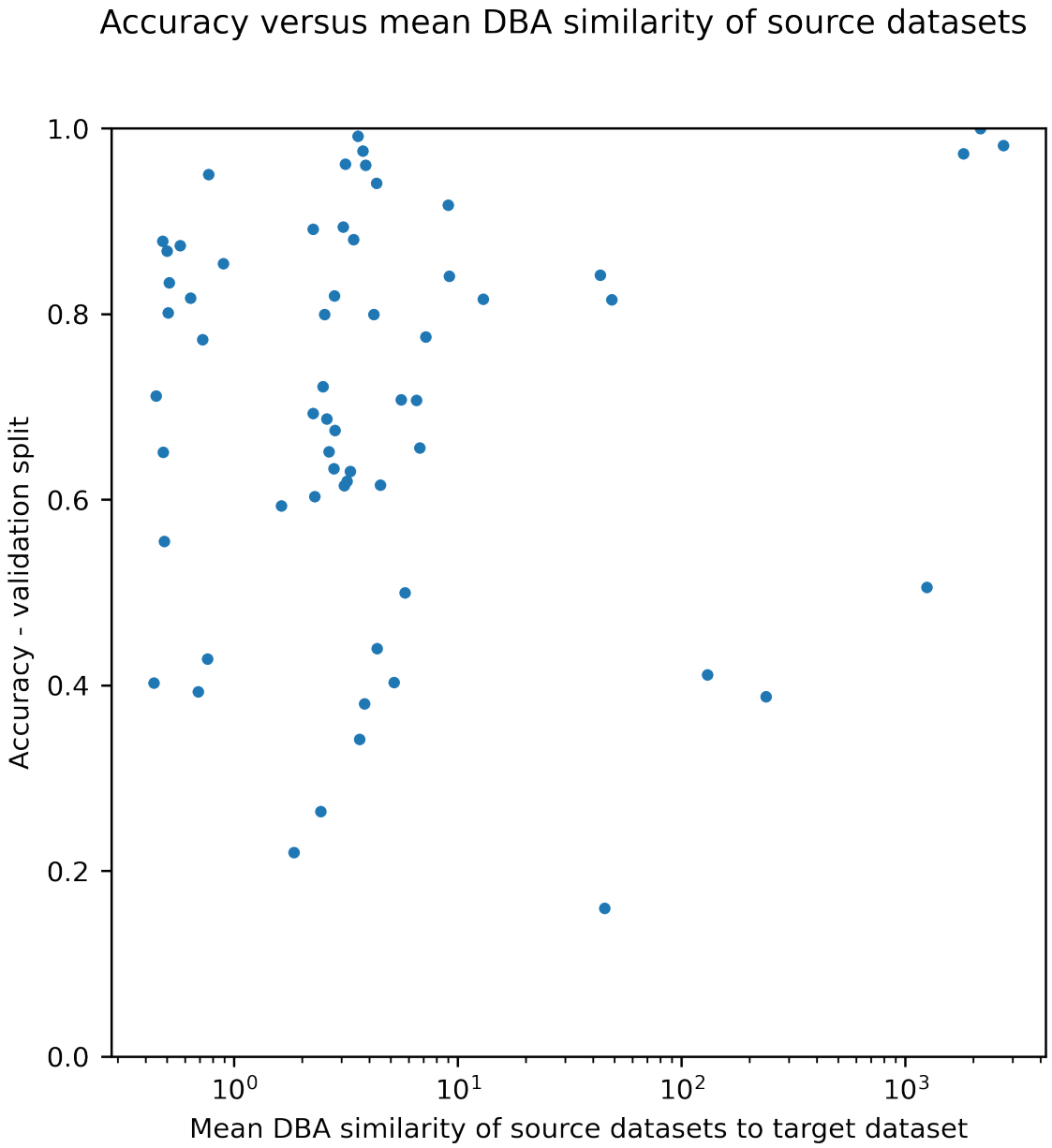

Figure 4.11:

4.6. Influence of accuracy of the components used in the ensemble on the target classifiers' accuracy

This short section checks the hypothesis that the better the model we use in *ensemble* approach, the better the end results on the target classifier. As we see on the plot 4.12, which shows the mean accuracy of components (measured on each source dataset, using test split) versus the accuracy of the target classifier (measured on the test split of target dataset). The correlation between the two statistics is very moderate (0.15).

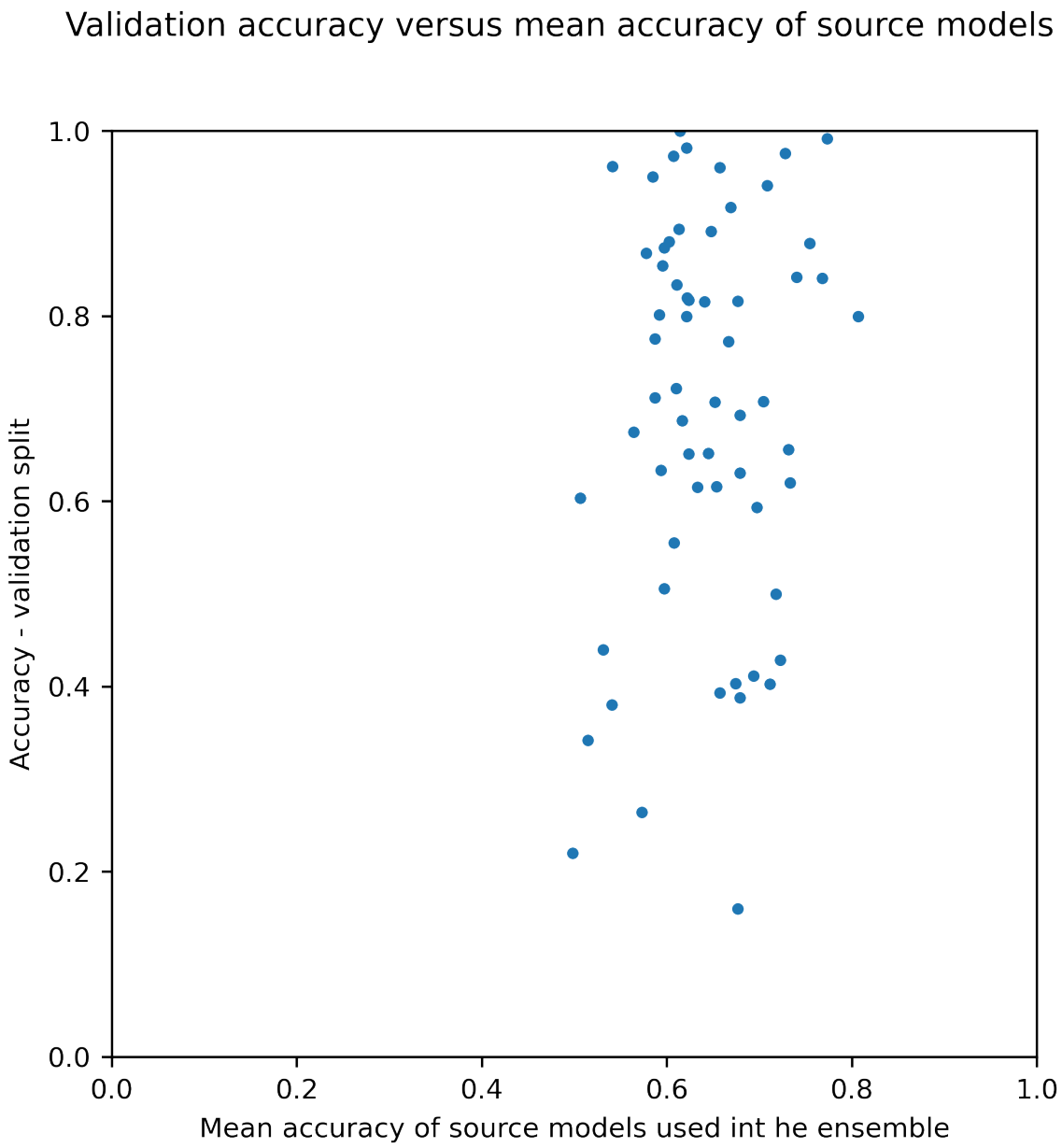

Figure 4.12: The weak influence of the accuracy of classifiers used in the *ensemble* method on the target classifier.

4.7. Number of datasets used in multi-source transfer learning

For the previous experiments, we used 5 source datasets, both for the *ensemble* and *baseline* approach. However, the number of experiments is arbitrary. This thesis aims to explore the benefits of multi-source transfer learning, compared to single-source transfer learning described in [6]. Similarly as in the previous sections we compared *ensemble* and *baseline* approaches to models trained without transfer learning, we compared the accuracy and loss for models pre-

trained with 3, 5, and 8 source datasets.

We found that all configurations give good results while using 5 datasets gives the best results. The averaged accuracy and loss plot is shown in figure 4.13. We used the ensemble approach for this comparison and we used the DBA similarity measure.

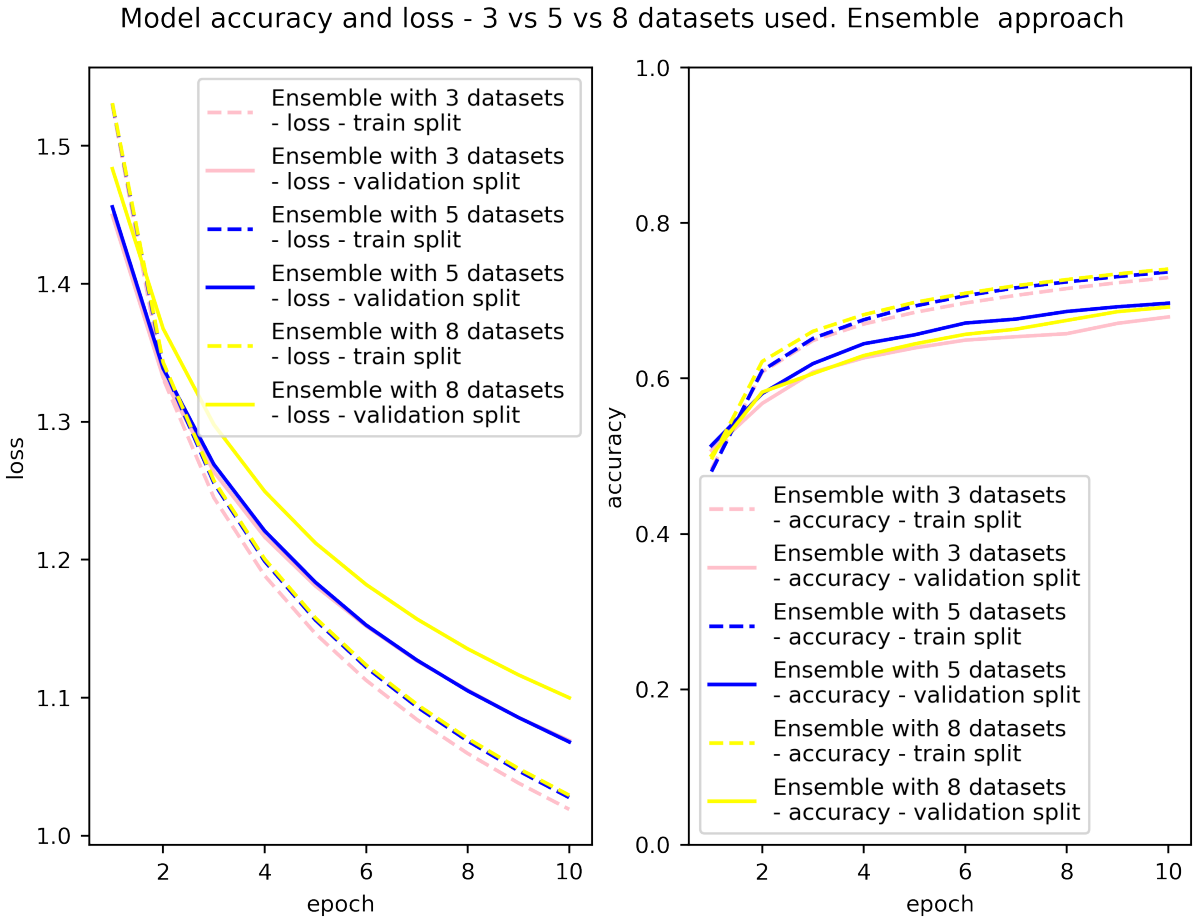

Figure 4.13: Accuracy and loss on each epoch, averaged over several target datasets

Figures 4.14 and 4.15 also presents the win tie loss diagram comparing the same experiment with 3 and 5 datasets, and with 5 and 8 datasets.

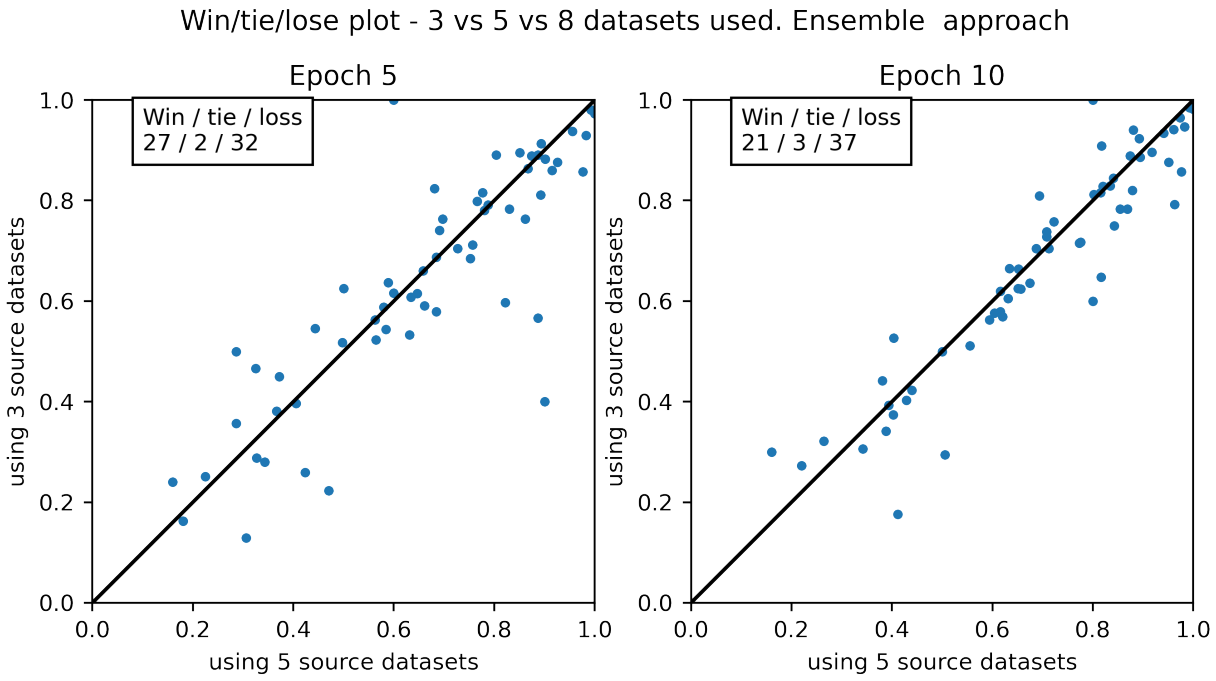

Figure 4.14: Win / tie / loss diagram comparing the ensemble approach with 3 and 5 datasets (DBA similarity measure is used for choosing the source datasets

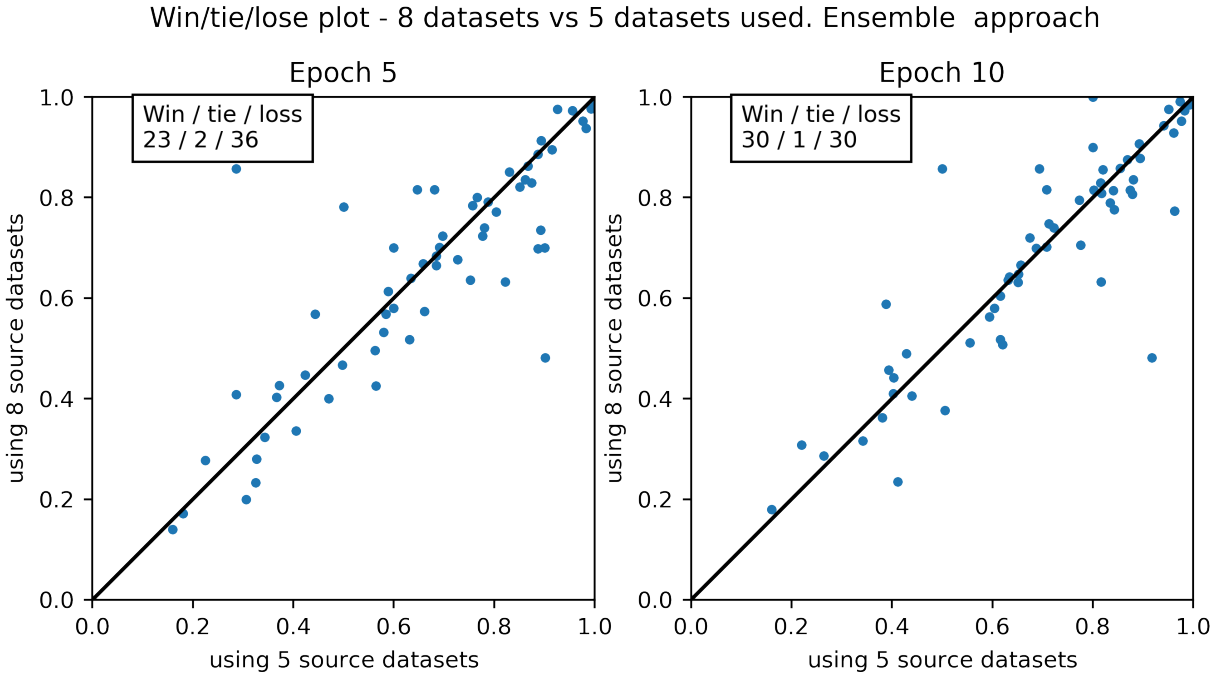

Figure 4.15: Win / tie / loss diagram comparing the ensemble approach with 5 and 8 datasets (DBA similarity measure is used for choosing the source datasets

5. Conclusions

In this thesis, we extended the research on transfer learning in time series classification. We proposed two approaches to multi-source transfer learning: the *baseline* and *ensemble* approach, the second inspired by the success of ensemble classifiers in the time series classification domain. The *baseline* approach can be summarized as pre-training a neural network on a concatenated dataset consisting of multiple datasets, and then using the weights from the first layers to initialize the final classifier. In the *ensemble* approach, we train several models, each on a different source dataset, and then we construct an ensemble of those methods initialized with the weights obtained from pre-training. We have proven that for both approaches, multi-source transfer learning increased the accuracy of the classifiers. The usage of transfer learning also mitigated overfitting, which is very common for small datasets present in the time series classification domain. Furthermore, a detailed comparative analysis of the two approaches was conducted.

The *ensemble* approach proved to be better in terms of performance results. However, the size of the ensemble model may be a limitation, especially with larger networks, bigger datasets, or in environments with limited memory. Furthermore, the *ensemble* approach is slightly slower, as we train multiple classifiers and the resulting model is bigger. However, training the component models for the ensemble can be parallelized and the components can be reused and time can be saved.

We also considered two methods of selecting the datasets for pre-training the source classifiers. The method based on the DBA similarity measure has proven to be slightly advantageous compared to picking the datasets randomly or by trial and error. We believe that this method may be improved by enhancing the similarity measure.

We also experimented with the number of source datasets. Experiments were conducted using 3, 5, and 8, using the *ensemble* approach and All of those settings resulted in decent results, and using 5 datasets occurred to be averagely more effective than using 3 or 5 datasets.

5.1. Further works

As an extension to the work done for the purpose of this thesis, we may consider an extension to the source dataset selection algorithm. We may consider measuring the similarity of shapelets, as those are the key attributes learned by convolutional layers.

Also, architectures other than Fully Convolutional Network could be used in the future. Both *ensemble* and *baseline* approaches are generally applicable to deep neural networks. As we focused on the transfer learning aspect and the computational resources were limited, we tested only the Fully Convolutional Network architecture. It may be worth considering performing the same analysis but using a fixed input size network, long short-term memory network, or residual networks.

The study may also be extended to testing datasets outside of the UCR time series archive, or testing transfer learning between different categories in the UCR archive, which wasn't planned in the scope of this thesis due to computational limitations.

The number of datasets used for both *ensemble* or *baseline* approaches can be further studied. We can consider numbers other than 3, 5, and 8 (e.g. using all datasets available in the UCR archive). The effort could be combined with improving the similarity measure, in order to develop a method to obtain the optimal number of datasets given the similarity measure, e.g. with stepwise selection of datasets similar as in feature selection.

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.