Knowledge-Based Systems 239 (2022) 107976

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Intra-domain and cross-domain transfer learning for time series data—How transferable are the features?

Erik Otović [a], Marko Njirjak [a], Dario Jozinović [b,c], Goran Mauša [a,d], Alberto Michelini [b], Ivan Štajduhar [a,d,*]

[a] Department of Computer Engineering, Faculty of Engineering, University of Rijeka, Rijeka, Croatia
[b] Istituto Nazionale di Geofisica e Vulcanologia, Rome, Italy
[c] Department of Science, Roma Tre University, Rome, Italy
[d] Center for Artificial Intelligence and Cybersecurity, University of Rijeka, Rijeka, Croatia

ARTICLE INFO

Article history:
Received 22 August 2021
Received in revised form 7 November 2021
Accepted 13 December 2021
Available online 24 December 2021

Keywords:
Machine learning
Transfer learning
Time series
Fine-tuning
Convolutional neural networks

ABSTRACT

In practice, it is very challenging and sometimes impossible to collect datasets of labelled data large enough to successfully train a machine learning model, and one possible solution to this problem is using transfer learning. In this study, we investigate how transferable are features between different domains of time series data and under what conditions. The effects of transfer learning are observed in terms of the predictive performance of the models and their convergence rate during training. In our experiment, we used reduced datasets of 1500 and 9000 data instances to mimic real-world conditions. We trained two sets of models (four different architectures) on the reduced datasets: those trained with transfer learning and those trained from scratch. Knowledge transfer was performed both within the same application domain (seismology) and between different application domains (seismology, speech, medicine, finance). We observed the prediction performance of the models and their training convergence rate. We repeated the experiments seven times and applied statistical tests to confirm the validity of the results. The overall conclusion of our study is that transfer learning is highly likely to either increase or not negatively affect the model's predictive performance or its training convergence rate. We discuss which source and target domains are compatible for knowledge transfer. We also discuss the effect of the target dataset size and the choice of the model and its hyperparameters on transfer learning.

1. Introduction

In recent years, deep learning techniques have become increasingly popular and have brought new and exciting challenges. One of the major challenges and obstacles in training deep neural networks is the need for datasets that contain sufficient amounts of training instances. Creating such datasets is generally time-consuming, which can slow down the application of deep learning in some domains. For example, this may be the case when it is difficult to collect additional data instances because the observed phenomenon is very rare, or labelling instances for supervised learning is time-consuming because it must be done manually. Transfer learning (TL) is one of the possible approaches to combat these problems.

TL allows a machine learning (ML) model trained to solve one problem to be adapted or fine-tuned to solve another problem. In this way, some of the knowledge contained within the model from the first task is used to solve the second task. Knowledge transfer reduces the number of training instances required to solve another task compared to training with randomly initialised models, reduces training time, and leads to better accuracy. One of the domains where this approach has proven useful is image classification. There are several state-of-the-art models (such as VGG or Inception) pretrained on large image datasets that can be fine-tuned to solve other problems with a much smaller dataset and in much less time (see Review [1]). In this context, TL has enabled the application of these architectures to problems where they could not otherwise be (successfully) applied due to the small amount of training data or due to computationally intensive or lengthy computational operations involved in training models using large datasets.

1.1. Related work

In recent years, some research papers have been published reporting the application of TL for time series (TS) classification

* Corresponding author at: Department of Computer Engineering, Faculty of Engineering, University of Rijeka, Rijeka, Croatia.
E-mail address: istajduh@riteh.hr (I. Štajduhar).

https://doi.org/10.1016/j.knosys.2021.107976
0950-7051/

and prediction. However, the total number of reported papers is still very limited. The works on this topic can be divided into two categories. The works in the first category tend to explain a particular situation in which TL was used. Such works can give the reader an idea of good practices in knowledge transfer. Most of these works are focused on medicine. For example, in [2], TL knowledge about one person's EEG signals was transferred to another person's case for an emotion classification task. In [3], the authors presented a TS method for anomaly detection that can improve the performance of automated monitoring systems (e.g. in hospital care). There are also examples of TS predictions improved by TL. In [4], the authors attempt to predict wind speed at a new site in the short term by transferring knowledge of wind speed from data-rich sites. In [5], the authors investigate the use of TL for earthquake ground motion prediction using multi-station seismic TS by transferring knowledge from two different seismological datasets with models trained for the same problem or models trained for a different (seismological) problem.

The works from the second category tend to evolve and popularise TL for TS data in the same way it has been popularised for image classification. These works propose new pretrained models and TL frameworks for TS data. In [6], the authors propose a ConvTimeNet model pretrained and validated on a UCR dataset. In [7], the authors analyse TS data whose properties change over time and explore how to exploit knowledge gained earlier once the properties of the series change.

In [8], the authors studied TL for TS data classification using a fully convolutional neural network. They used datasets from different domains obtained from the UCR archive and tested pairs of datasets—one as the source dataset and another as the target dataset. However, most of these datasets contain several hundred training instances or less, which could limit the knowledge that the model acquires during initial learning. This poses a potential problem since the usefulness of TL depends on how well the pretrained model has been trained.

All these studies were limited by some parameter: either to one architecture, or one application domain, or only one dataset, etc., so we cannot look at the effects in general and draw general conclusions about TL for TS data. As far as we know, there is no study that has systematically observed the effects of TL in the domain of TS in a broader sense.

1.2. Research objectives

This paper attempts to fill the gap we described previously through a series of experiments. In order to get a broader picture of TL, the experiments and tests conducted focus on the following ideas:

1. It is reasonable to expect that knowledge transfer between related domains might be more beneficial than knowledge transfer between unrelated domains. For the purpose of this study, we consider two TS data to be related if they were sampled from the same type of underlying process (their source), using the same propagation medium, and if they were recorded using the same type of instrument. These factors were considered because they affect the amplitude, frequency spectra and temporal patterns — which are important characteristics of TS data in this study. This paper tests knowledge transfer within the same domain (intra-domain TL) and knowledge transfer between different domains (cross-domain TL). In the case of intra-domain TL, we test knowledge transfer between seismic datasets that differ in epicentral distances (local to teleseismic), sampling rates (20 Hz and 100 Hz) and different units (digital counts and m/s). The application of ML in seismology is still at an early stage with the datasets that are often small in size due to e.g. being focused on seismicity in certain geographic areas, having a small number of large earthquakes due to Gutenberg–Richter law, etc. Our conclusions could be applied in such cases in order to obtain better predictive performance. In the case of cross-domain TL, we test knowledge transfer between the domains of seismology, sound, medicine, and finance in which TS data stem from different processes (earthquakes, human speech, muscle activity, stock market trading). All of these varieties in datasets influence the signal in terms of its amplitude and frequency spectra and temporal patterns. The reason why cross-domain TL is interesting is that all TS data can be decomposed into a linear combination of sine and cosine waves, and in this way the TS data from different domains are related to some extent. This suggests that there is shared knowledge that could benefit the training process of the ML models when applied to two domains that do not seem to be related at all in the real world.
2. We use several datasets to perform experiments related to intra-domain and cross-domain TL. Some of these datasets concern classification tasks, while others deal with regression tasks. We ensure that the datasets are of appropriate size to pretrain the models. To get a better insight into how TL works, we test the knowledge transfer between all possible pairs of datasets used. Here, a different number of channels between source and target datasets poses a problem. We present our solution to this problem before proceeding with TL.
3. As mentioned earlier, learning by knowledge transfer allows training models using smaller datasets. However, it is well known that the quality of the ML model strongly depends on the size of the training set. Therefore, in our experiments, we created two variants with different sizes for each training dataset. This should allow us to see how the effects of TL change with the size of the training set.
4. Not all models are equally well suited to all tasks. Therefore, we expect that some models trained using TL might perform better than others. Since the goal of this paper is to get a broader impression of TL, four ML models were used for all experiments: two models are from seismology, and the other two models are general purpose models for TS data. In this way, our conclusions are not tied to any particular model.
5. The success of the knowledge transfer depends on the hyperparameters chosen to control the training process on the target dataset (i.e. fine-tuning). In our case, we have only one such parameter, and that is the learning rate multiplier, which controls how fast the weights of the pretrained layers are changed during fine-tuning. We assume that different values of this parameter are appropriate for different cases. Therefore, we perform a grid search to determine the optimal value of this parameter for each case. In this way, the best possible results are obtained in all cases and thus the influence of this parameter on our conclusions is minimised.
6. In model training, the main focus is on achieving the best possible performance. However, in other domains, such as image recognition, it has been observed that pretrained models converge faster. Therefore, in this work, we study the performance of the models and the rate of their convergence.

In this sense, this work is systematic and comprehensive because it attempts to explore all possible combinations of given models, hyperparameters, datasets, and training set sizes in order

to eliminate the need to make assumptions that might later bias the results. In this way, we overcome the limitations imposed in other studies and provide greater insight into the implications of TL for TS data.

All source code required to replicate these results has been published and is freely available for download, along with the results obtained, from the GitHub repository.[1]

2. Materials and methods

In this section, we begin by describing the datasets and models used. Next, we explain the TL process, the obstacles we faced, and how we solved them. Finally, we describe all parts of our experiment: data preparation, changes to the model, training procedures, evaluation metrics, and statistical tests. We also provide information about the software and hardware we used.

2.1. Datasets

In this section, we present the TS datasets used in our experiments. Three of them are seismic datasets (LOMAX, LEN-DB and STEAD), one is a spoken word dataset (acoustic signals), one is a medical dataset (EMG) and the last one is a stock market price dataset (S&P 500). The focus of our study was on TS TL across different domains, but also within the same domain (seismology), which explains the inclusion of several seismic datasets.

2.1.1. Lomax dataset

This seismological dataset [9] was presented and used in [10]. In this paper, we refer to it as the LOMAX dataset because it has not been named. This dataset contains 22,046 three-channel (BHZ/N/E) seismograms of global earthquakes at any epicentral distance collected using the MedNet station network from 2010 to 2018. From the same station network, 13,009 noise seismograms were collected and provided in the dataset. All recorded seismograms have a duration of 50 s (1001 sampling points) and were acquired at a sampling rate of 20 Hz. The units of the seismograms in this dataset are metres per second.

The earthquake waveforms begin 5 s before the arrival of the first P wave. As a form of quality control, [10] kept only the earthquake waveforms with a signal-to-noise ratio (SNR) greater than 3.0 and the noise waveforms with an SNR less than 1.5.

In [10] the dataset was preprocessed so that each instance of the seismogram was normalised separately. The maximum value of each waveform signal from all three channels is stored (they called it *stream max*), and this value is later used in the neural network with the aim of improving the results.

Since we were using the data for the task of determining earthquake magnitude, we decided to filter the test set to include only those instances that were correctly classified by the convolutional neural network (CNN) model used in the original study (87% accuracy). We also visually inspected the misclassified earthquakes (i.e. those that the CNN model classified as seismic noise) and found that most of them were earthquake signals buried in the seismic noise and not easily recognised as earthquakes. This confirmed our decision to omit these data when training the model to determine earthquake magnitude. The dataset also contains some duplicates, which we removed. The total size of the resulting dataset containing only the earthquakes was 19,426.

[1] https://github.com/ecokeco/tstl.

2.1.2. LEN-DB dataset

LEN-DB dataset was published in 2020 with the aim of collecting a sufficiently large amount of data for use with ML [11]. It is a global library of local earthquakes created by collecting data from 1,487 broadband or very broadband monitoring stations distributed around the world. The entire dataset is publicly available online in the form of a single HDF5 file [12]. The dataset consists of 1, 249, 411 three-channel seismograms, of which 631,105 seismograms contain earthquakes and 618,306 contain seismic noise. Seismograms containing earthquakes were obtained from 304,878 different earthquakes. Seismograms are 27 s long, with a sampling frequency of 20 Hz, which means that one channel of a seismogram contains 540 sampling points. The ground motion in the seismograms of this dataset is given in metres per second. We kept only the seismograms that contain earthquakes because we are performing a task of earthquake magnitude determination.

In their paper, the authors also presented a simple example of using the dataset for earthquake detection. The model used in the example was a variant of the model used in [10]. The input data in their paper was normalised and a *stream max* value was determined as was done in [10].

2.1.3. STanford EArtquake Dataset

Stanford EArtquake Dataset (or STEAD for short) is a database of seismograms of local earthquakes collected from stations around the world [13]. Thus, this dataset deals with the same phenomena as LEN-DB. The entire dataset is publicly available as an HDF5 file. The dataset file downloaded from the official Github repository is called *stead_waveforms_11_13_19.hdf5*. The file was downloaded on 2020-05-03, and the Github repository has been updated several times since then. Using the repository history, it is possible to see the repository as it looked on the day we downloaded the file and download the same file we downloaded.

This dataset consists of 1, 137, 793 three-channel seismograms, of which 1, 031, 908 represent earthquakes and 105,885 represent noise. All seismograms have a sampling frequency of 100 Hz and a time duration of 60 s, resulting in 6000 sampling points per seismogram channel. The units of the seismograms are counts that depend on the transfer function of the device recording the waveform. For this reason, two seismograms of the same ground motion recorded with different instruments may give different signal amplitudes. Therefore, seismograms recorded with different instruments are not directly comparable.

2.1.4. Speech commands dataset

Speech Commands is a dataset compiled and published by Google Brain [14]. In this paper, it is referred to by the abbreviation SPEECH. This dataset consists of approximately 105,000 WAV files containing the sound of the recorded word, and the sampling frequency used is 16 kHz. All recordings of spoken words are classified into one of 35 possible classes, and in addition there are some longer recordings of sounds that do not contain speech. We have empirically found that the ML models described in Section 2.2 have the best performance when we resample the audio to 8 kHz. This is especially true for the ConvNetQuake INGV model, which failed to converge on the original recordings at 16 kHz sampling frequency. The solution to this problem is described in more detail in Section 2.2.1. Each audio track lasts about 1 second, and in preprocessing we reduce all audio tracks to the same duration of exactly 1 second (8000 sampling points). We do this by concatenating the recordings that last less than 1 second with the noise contained in the dataset. For recordings longer than 1 s, we keep the first second (8000 sampling points) of audio and discard the rest.

At the time of writing, there are two versions of the dataset, and information about them and instructions on where to download them can be found in the corresponding paper [14]. In our

experiments, we used version v0.02, which contains more different words (i.e. classes) and is more numerous than the previous version.

2.1.5. EMG dataset

We have acquired the EMG (Electromyography) dataset from [15]. Detailed information about the dataset can be found in [16]. An EMG signal is a biomedical signal that represents the electrical activity produced by muscles when they are stimulated. This dataset was created by recording an EMG signal using the Myo armband. The armband consists of 8 evenly spaced non-invasive sensors that record a signal at a sampling rate of 200 Hz. The application of this dataset is in the detection of hand movements using ML methods.

The dataset consists of two subsets called Pinch and Roshambo. The only difference between these two subsets is in the motions executed. In the case of the Pinch subset, there are four classes representing pinches between the thumb and index, middle, ring and little finger. The Roshambo subset contains three movements (stone, paper, scissors), about which more information can be found in the accompanying paper. The Pinch subset is more numerous and for this reason we decided to use it exclusively. The movements included in it were performed by 22 participants. Each participant performed three sessions in which each gesture was performed five times for two seconds. Between the two movements there is a relaxation period of one second.

This dataset contains an eight-channel signal. Using a sliding window that moves through the recorded signals, we obtain samples of equal length, making sure that there is only one type of motion within the sliding window. We started with the settings used in [17], i.e. we set the length of the sliding window to 200 ms and shifted it by 100 ms in each iteration. However, we empirically found that our models performed significantly better when we used 400 ms (80 sampling points) long sliding windows and a 50 ms step. In the end, this method resulted in a dataset consisting of 32,438 instances.

In [18] the authors have shown that it is possible to obtain good results with a smaller number of channels. Therefore, we have kept only three channels, since all other datasets are single- or three-channel. The different number of channels between datasets is a problem in TL. In this way, only the problems of TL between single-channel and three-channel cases need to be addressed. Our solution to this problem is explained in more detail in Section 2.3.

Preserved channels have the indices 0, 2, and 5. Since the sensors are arranged circularly around the arm, we chose these channels so that the distance between the corresponding sensors is approximately equal.

2.1.6. S&P 500 dataset

In reviewing existing work in the field of stock price forecasting, we found that it is a common choice to use the S&P 500 (Standard & Poor's 500) dataset [19,20]. In this paper, we will refer to it by the abbreviation S&P 500. This dataset is publicly available through Yahoo finance[2] and can be downloaded for any time period. In [19], the authors argue that S&P 500 is more stable than individual company stock prices, which is a better potential for the predictive model. For the purposes of this paper, we used data from December 30, 1927 to November 26, 2018 (inclusive).

The data is contained in a single CSV file that contains the following information: Date, Open, High, Low, Close, Adjusted Close and Volume on the given date. When looking at the data, you may notice that records are missing for specific days. This is because the exchange was closed on those days. Since prices could not change on those days, we treat the data as a continuous TS with no missing values.

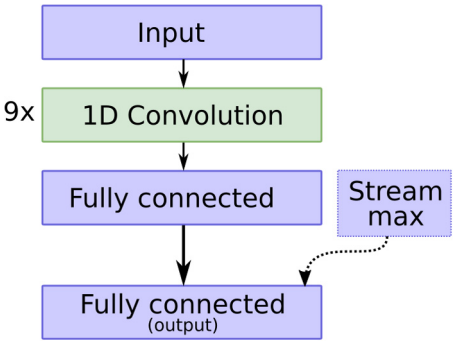

Fig. 1. A succinct schematic of ConvNetQuake INGV architecture.

We adopted the data extraction method and stock price prediction experiment from [19]. First, we extract only the closing price from the downloaded data and then divide this time sequence into smaller parts using a sliding window. The sliding window consists of a consecutive sequence of 50 closing prices and advances by a single record at each step, starting from the oldest date to the most recent. The result of this operation is a set of 22,681 TS of length 50, representing a closing price in 50 consecutive workdays. We also associate with each obtained series of 50 records the closing price that was valid on the 51st day. The objective of the ML model is to predict the price on the 51st day based on the past 50 consecutive records.

Since we are dealing with TS data, we paid special attention to preprocessing to avoid data leakage between training, validation and test sets. We divide the dataset into training, validation and test sets in such a way that starting from the oldest to the most recent date, we take 70% of the records for the training set, the next 15% for the validation set and the last 15% for the test set. Each set is created by a sliding window over the records associated with it. In this way, the model cannot come into contact with the test data in any way during training, and the test data is only used to evaluate the performance of the model at the very end, ensuring that no data leakage occurs between the sets. Once the data were extracted, they were processed as described in Section 2.4.1.

An overview of the previously described datasets is given in Table 1.

2.2. ML models

In this section, we briefly describe the models we chose for our experiments. Seismology-specific models, ConvNetQuake INGV and MagNet, are described in Sections 2.2.1 and 2.2.2, respectively. General purpose models for TS data, MLSTM FCN and TCN, are described in Sections 2.2.3 and 2.2.4, respectively.

For each model, we give basic information and the reference to the paper from which it was taken. Hyperparameters not listed here, such as weight initialisation methods or dropout rates, were kept as they were in the original work. Therefore, all changes in the model architecture and the changed hyperparameter values are reported and justified in the following sections.

2.2.1. ConvNetQuake INGV

The work in [10] introduced ConvNetQuake INGV (shown in Fig. 1), an adaptation of ConvNetQuake [21], a CNN model for detecting and determining magnitude, location, and depth of global earthquakes at arbitrary distances (from local to far-teleseismic)

[2] https://finance.yahoo.com/quote/%5EGSPC.

Table 1
An overview of the chosen datasets and their characteristics.

Dataset	Task type	Domain	Size	Channels	Sampling frequency	Sampling points per channel	Waveform duration
LOMAX [9]	Regression	Seismology	19,426	3	20 Hz	1,001	50 s
LEN-DB [11]	Regression	Seismology	629,096	3	20 Hz	540	27 s
STEAD [13]	Regression	Seismology	1,031,908	3	100 Hz	6,000	60 s
SPEECH [14]	Classification	Audio	105,829	1	16 kHz (resampled to 8 kHz)	16,000 (8,000 after resampling)	1 s
EMG [15]	Classification	Medicine	32,438	3	200 Hz	80	0.4 s
S&P 500 [19,20]	Regression	Finances	22,681	1	Once every working day	50	50 days

over a wide range of magnitudes using raw single-station waveforms.

In their experiment, [10] used the LOMAX dataset. Information about the distance, azimuth, depth, and magnitude of the earthquake was available as event metadata. In addition, a parameter (*stream_max*) representing the maximum absolute value in a given waveform over all three channels was calculated for each stream of input data. The main purpose of this parameter is to assist in estimating the CNN magnitude when the normalised waveform is input. In their work, the model performed discrete prediction of distance, azimuth, depth, and magnitude into the bins, where each bin represented a range of values. To overcome the problem of overfitting, the authors introduced L2 regularisation of 0.001 into the convolutional layers.

This architecture has nine 1D convolutional layers whose job is to extract features from the input signal. This is followed by two fully connected layers. The outputs of the last convolutional layer are passed along a single *stream_max* value to the input of the first fully connected layer. The output layer was adapted to our needs. In our case, the output node contained only a single output neuron for earthquake magnitude estimation. This was necessary because the original model only performed discrete prediction of magnitude, while we needed continuous output.

We found that this model did not always converge on the SPEECH, EMG and S&P 500 datasets. Therefore, we had to create three variants of this model. Each variant had the same architecture as the original, but with different values for L2 regularisation and initialisation of the convolutional layers. For the SPEECH dataset, it was sufficient to use Tensorflow's *he_normal* initialiser for the convolutional layers and resample the audio files to 8 kHz. In the variant intended for the EMG dataset, we changed the initialiser to Tensorflow's *glorot_normal* and removed the L2 regularisation completely. In order for the model to converge on the S&P 500 dataset, it was necessary to increase the L2 regularisation to 0.7 and use Tensorflow's *he_normal* to initialise the convolutional layers.

2.2.2. MagNet

The MagNet model (shown in Fig. 2) is a deep learning regression model for end-to-end prediction of earthquake magnitude [22]. The model predicts magnitude from the raw waveforms of a single station. The authors claim that the model is insensitive to data normalisation, so non-normalised waveforms can be used as inputs. The significance of the MagNet model lies in the fact that it is the first deep-learning approach that has successfully estimated magnitudes from raw seismic waveform signals from a single station.

This architecture takes a three-channel waveform as input, which is followed by two convolutional layers that do not use an activation function, and whose two main tasks are to reduce the dimensionality of the data and to extract the features. Each convolutional layer is followed by a dropout layer with a dropout factor of 0.2 and a max-pooling layer. The task of the dropout

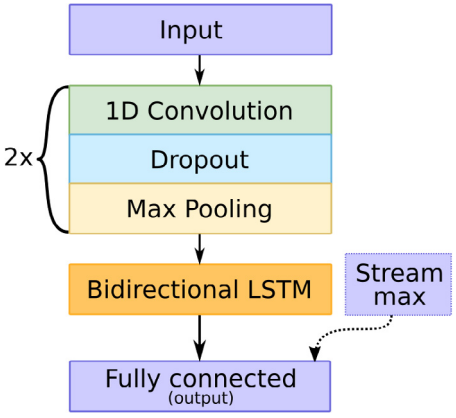

Fig. 2. A succinct schematic of MagNet architecture.

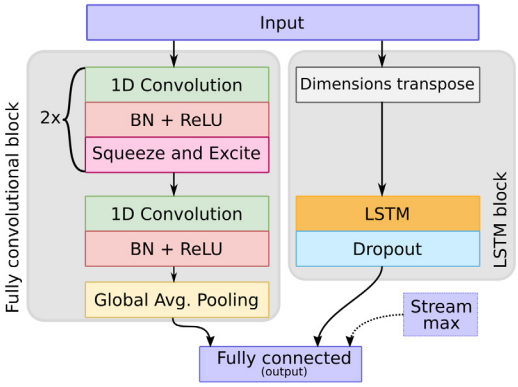

Fig. 3. A succinct schematic of MLSTM FCN architecture.

layer is to achieve regularisation and each max-pooling layer reduces the data four times with the aim of reducing the training time. This is followed by a bidirectional LSTM layer with 100 units. [22] claims that most of the learning occurs in the LSTM units and that they are a suitable tool for modelling TS data, such as earthquakes. Ultimately, the output of the network is a single neuron with no activation function (i.e. a linear response neuron). No changes were made to this architecture, as it was perfectly adequate for our experiment.

2.2.3. MLSTM FCN

In 2019, the Multivariate LSTM Fully Convolutional Network (MLSTM FCN) architecture (shown in Fig. 3) was proposed, which performed better than state-of-the-art models in classifying complex multivariate TS data [23]. This architecture is a generalisation of the LSTM FCN architecture, which is only suitable for univariate signals.

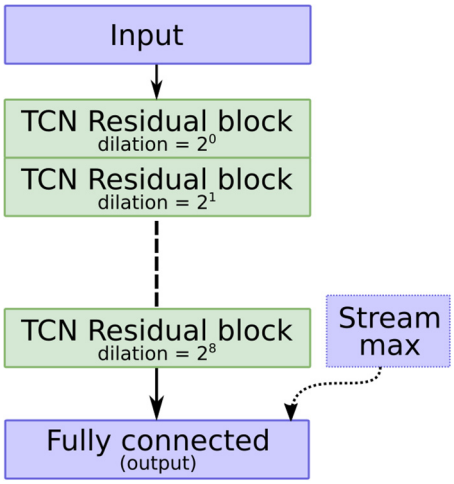

Fig. 4. A succinct schematic of TCN architecture.

In [23], MLSTM FCN was tested on 35 datasets originating from three different sources and obtained state-of-the-art results in 28 cases. The datasets used were from different domains, such as medical care, speech and speaker recognition, and activity recognition. The tasks it was tested on varied widely. For example, the smallest number of channels within the task was two, while the largest was 570. In terms of the TS length, the shortest length was 15 and the longest was 5396. It can be seen that all the data we used for the experiment are comparable in size to those on which MLSTM FCN was tested. For this reason, we decided to leave the architecture as it is without intervening.

In general, the model consists of two parallel branches called the fully convolutional block and the LSTM block. The inputs of both branches are fed with the same data and the outputs of these branches are finally combined and based on that a final prediction is made. A fully convolutional block consists of three temporal convolutional blocks. Each temporal convolutional block consists of a convolutional layer, batch normalisation, followed by a rectified linear unit (ReLU) activation function. In addition, the first two temporal blocks contain a squeeze-and-excite block that adaptively recalibrates the input feature maps. The output of the last temporal block is brought to the global average pooling layer, ending the first branch. In the second branch, the first step is to transpose the temporal dimensions. This means that the input sequence containing M channels and Q time steps is converted to a sequence of Q channels and M time steps. This causes the LSTM block to process all data in M steps instead of Q steps. Of course, this is only beneficial when M is smaller than Q. The authors show in their paper that this significantly reduces the training time without significantly affecting the accuracy of the model. The temporally-transposed data is fed to a plain LSTM layer, followed by a dropout layer to prevent overfitting. A high dropout rate of 80% is used.

2.2.4. TCN

In their recent paper [24], the authors presented the results of their research comparing generic convolutional networks and recurrent sequence modelling networks. In their experiment, they first presented a temporal CNN and then compared its performance on various sequence modelling tasks with the performance of long short-term memory (LSTM), gated recurrent unit (GRU), and vanilla recurrent neural network (RNN) architectures. They explain how TCN provides a simple but powerful starting point for sequence modelling and how it achieves better results than generic recurrent architectures.

The main features of the TCN architecture are: (1) the convolutions used are causal, which means that information from the future is not present in the past, and (2) the architecture maps the input sequence of arbitrary length into an output sequence of the same length as the RNN. This architecture consists of residual blocks that are sequentially connected. Within each residual block, there is a sequence of layers: dilated causal convolution, weight normalisation, ReLU activation, and dropout, and this sequence is repeated twice. The output of the residual block is obtained by summing the inputs of the residual block and the outputs of the last dropout layer. Dilated convolutions allow an efficient increase of the receptive field of the network, which the authors believe is of great importance for modelling time sequences with TCN. Therefore, the receptive field of TCN can be increased by increasing the number of convolutional layers, using a larger kernel size or a larger dilation factor. The performance of TCN and generic recurrent architectures was compared on 11 tasks, which are mainly used to compare the performance of recurrent networks. When the authors performed the tests, the architecture used for each task was the same, but with different hyperparameters. The following hyperparameters were varied: the size of the convolution kernel, the number of filters, and the depth of the network to control the size of the receptive field.

The choice of hyperparameters for TCN directly affects the receptive field of the model. The receptive field depends directly on the length of the TS data to which the model is applied. As the authors have noted, problems can arise when performing TL because source and target tasks can have very different numbers of time points. This problem may lead to poor TCN performance after TL. Namely, different application domains may require different lengths of time (i.e. perceptive field) for the model to predict the outcome. This is exactly the case in this work — the S&P 500 dataset has the fewest time points (only 50), while the SPEECH dataset has the most time points (as many as 8000). In the original paper, the authors tested the model over different lengths of TS data and offered the hyperparameters used. For the purposes of this paper, hyperparameters were used for time sequences that contained 600 time points. These hyperparameters correspond to our median case. In this way, there are three datasets that have less than 600 time points (S&P 500, EMG, and LEN-DB) and three datasets that have more than 600 time points (LOMAX, STEAD, and SPEECH).

For reproducibility, these hyperparameters were taken from "The adding problem", namely: kernel size = 8, number of filters = 24, dilation = 8, dropout = 0.0, and gradient clip = N/A.

ML model obtained by these hyperparameters is depicted in Fig. 4.

2.3. Transfer learning

TL [25] is a technique that attempts to improve the model's performance on the target task T_t in domain D_t by using previously acquired knowledge from domain D_s and task T_s, where $D_s \neq D_t$ or $T_s \neq T_t$ (or both simultaneously). This procedure consists of two phases: In the first phase, the model is trained on the source task and in the second phase, the model is trained on the target task. In their research [26], the authors ask three questions that arise when using TL: "What to transfer", "How to transfer", and "When to transfer".

The questions "What to transfer" and "How to transfer" ask what part of the acquired knowledge from the source domain D_s can be useful in the target domain D_t and how this knowledge should be transferred. In our experiment, we try to exploit the fact that convolutional layers learn feature extraction by letting earlier layers learn simpler patterns and later layers learn more complex patterns [27]. Therefore, the model trained in the source

domain D_s retains the weights of the convolutional filters before moving to the target domain D_t to retain the acquired knowledge. How the model is adjusted during the domain transition and how the fine-tuning of the convolutional layers is done is described later in this paper.

The "When to transfer" question asks which source domains D_s lead to performance improvements in the target domain D_t. It is also possible to have the opposite effect, where a model pretrained in domain D_s performs worse in domain D_t; this phenomenon is called negative transfer. The answer to this question is not known for TS domains and is one of the research topics in this paper. To answer this question, we investigate the extent to which features learned in convolutional layers on one task are useful when transferred to the target task.

Two problems arise in the implementation of TL, which we describe below and explain how we solved them. The first problem is that the same architecture differs in its fully connected layers depending on which dataset it is applied to. More specifically, we use *stream max* (defined in Section 2.1.1) on some datasets and not on others. Also, depending on the nature of the task, there may be more or fewer output neurons. All selected regression problems involve the prediction of only one value (earthquake magnitude or stock price), so there is always one output neuron in their case. However, in classification problems, the number of output neurons depends solely on the number of predicted classes. We solved this problem by instantiating a new model suitable for the target task and copying the convolutional layer filters from the pretrained model to this model. As mentioned earlier, this procedure preserves the knowledge within the convolutional layers, while randomly initialising the weights in the fully connected layers.

Another problem is the different number of channels between datasets. The problem occurs when we want to fine-tune a model on a target dataset that has a different number of channels than the source dataset on which the model was originally trained. Specifically, the problem occurs in the first convolutional layer, whose number of channels depends on the number of channels in the dataset. Since all datasets used have either one or three channels, there are two problematic scenarios. The first occurs when the source dataset has one channel and the target dataset has three channels. We solved this problem by simply replicating the convolutional filters (and their weights) to obtain three channels in the first convolutional layer. This technique is used when the source dataset is SPEECH or S&P 500 and the target is a three-channel dataset.

The second problematic scenario occurs when the source is a three-channel dataset and the target is a single-channel dataset. More specifically, this occurs when the source dataset is LOMAX, LEN-DB, STEAD or EMG and the target dataset is SPEECH or S&P 500. We solved this problem by keeping the three-channel convolutional layers and copying a single channel in the target dataset to form a three-channel dataset. Note that the difference in signal length between the source and target datasets is not a problem, as the signal length does not affect the number of convolution filters in any way.

As mentioned earlier, the learned weights of the convolutional layers are retained and used instead of randomly initialising them on the target dataset before training. These convolutional layers are trained with learning rate α_{conv}, while the remaining layers are trained with learning rate α. The general assumption is that pretrained layers should be trained with different learning rates since they already encode some knowledge. α_{conv} is obtained by multiplying by the factor:

$$\alpha_{conv} = \omega \cdot \alpha \tag{1}$$

For the ω value, we selected 10 different values to examine. This is because we found that not all models achieve the maximum improvement at the same value of ω. Therefore, we performed a grid search, i.e. we ran the fine-tuning process 10 times, but each time with a different value of ω. For the values of ω, we used the values from the set {0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1.0, 1.25, 1.5, 2.0}. Although it is common practice to fine-tune pretrained layers with a lower learning rate, we have found through empirical experiments (Section 3.4) that using larger learning rates can also be beneficial in some cases.

2.4. Experimental setup and evaluation metrics

In this section we describe how we preprocessed the data, the modifications we made to the models, the hyperparameter values we used to run the experiment, the workflow of the experiment, and the metrics we used to evaluate the experiment.

2.4.1. Data preprocessing

Preprocessing and normalisation of the data is an important step because, depending on the method chosen, better or worse predictive accuracy of the model can be achieved and the time required for training can be either shortened or lengthened [28]. The method we applied to all datasets is described below, regardless of which method was used in the original papers. Before we could apply the chosen method, we had to randomly divide the datasets into training, validation, and test sets of 70%, 15%, and 15%, respectively. We introduce the following notation to explain the process of data preparation: S_{train} for the training set, S_{val} for the validation set, and S_{test} for the test set. Each of these subsets consists of a number of data instances, and each data instance consists of the pair (X_i, Y_i), where X_i represents the observations that are the input to the ML model, while Y_i represents the ground truth value that the model is trying to predict. An observation X_i is a matrix of dimensions $C \times N$, where C is the number of input channels and N is the number of sampling points recorded over time. The ith data instance belonging to the training set is denoted as $(X_{train,i}, Y_{train,i})$. Data instances in the validation and test sets are denoted analogously.

S_{train} is the first to be processed, and as a first step we find the minimum and maximum values for each input matrix $X_{train,i}$ and denote them by m_i and M_i, respectively. From these values, we can determine the corresponding *stream_max* value for the ith data instance by $stream\_max = max(|m|, |M|)$. Then, using the $min - max$ method, all values $x_{j,k}$ of the matrix $X_{train,i}$ are scaled to the range [0, 1] using the expression:

$$x'_{j,k} = \frac{x_{j,k} - m}{M - m} \tag{2}$$

The final step in preparing the training set is to centre each element of the matrix $X_{train,i}$ around zero by considering each position within the matrix separately by all training instances. An element at position (j, k) would be centred by subtracting from its value the mean of all elements at that position within the matrix $X_{train,i}$. Here j is the index of the currently observed channel ($j \in [1, C]$) and k is the index of the time point in the TS ($k \in [1, N]$). The average value for each pair (j, k) is obtained by dividing the sum of the values at position (j, k) over all $X_{train,i}$ by the number of data instances in the training set.

This process can also be performed using matrix operations, which improves performance and speeds up data preprocessing. First, we define a matrix of dimensions $C \times N$ containing the average value for each position (j, k) and call it a mask. Using matrix operations, it can be easily obtained as:

$$mask = \frac{\sum_{i=1}^{|S_{train}|} X_{train,i}}{|S_{train}|} \tag{3}$$

Then centring each value around zero can also be performed using the matrix operations $X'_{train,i} = X_{train,i} - mask$ for $i \in [1, |S_{train}|]$.

The preprocessing of S_{val} and S_{test} is identical to the preprocessing of S_{train}, except that the new mask is not computed, but the one computed during the preprocessing of S_{train} is used. Thus, we first determine *stream_max* for each instance, then scale each instance to the range $[0, 1]$ using the min–max method, and finally subtract the *mask* from each instance in the validation and test sets. Such data preparation methods (normalisation and standardisation) are necessary for faster and better convergence (unless the models are known to be insensitive to the statistical properties of the input values).

As target datasets, we use reduced variants of the above datasets in TL to simulate small-scale target datasets. This allows us to simulate a real-world situation where it is not possible to acquire a sufficiently large dataset, and to investigate how the size of the target training dataset affects the model. For each dataset mentioned, we create two reduced variants, which we call *1k5* and *9k*. We obtain these variants by reducing only the training sets to 1500 and 9000 instances, while leaving the test and validation sets unchanged. When reducing the training set, we randomly choose which data instances will be included in the reduced dataset, while maintaining the underlying distribution of classes (i.e. using stratification for classification datasets).

Since no resampling of the TS data is performed in the preprocessing step, the models perceive all the data as if they had been sampled at the same sampling frequency. This also affects the models' understanding of the frequency content of signals from different domains, since the perception of the content is affected by the sampling rate of the signal.

2.4.2. Model adjustments

In our experiment, there are four datasets bound to a regression problem (LOMAX, LEN-DB, STEAD, S&P 500), and two datasets bound to a classification problem (SPEECH and EMG). Therefore, we need to adjust each model to each task separately. We adjust the model by changing only the last fully connected layer. In the case of the regression problem, the last layer contains a single neuron since only a single value is predicted. In general, we could add as many neurons as values are predicted in the multi-target regression case. No activation function is applied to the last fully connected layer. In the case of a classification problem, we add as many neurons to the last layer as there are classes predicted using the softmax activation function.

Another modification of the model relates to the *stream max* input. This work is strongly inspired by the work presented in [10,29]. However, these two works have opposite views on the use of *stream max* input. However, this is likely a consequence of using different input data: in [10] they use the data in m/s, while the data used in [29] are not exempt from the instrument response (i.e., the data are in counts). Since information regarding the magnitude of the signal is lost during data preparation due to min–max scaling, this effectively means that the *stream max* input carries the important information in [10] and in [29] does not, since one needs to know the properties of the instrument (i.e. the instrument response) to correlate the count values with ground motion. We decided that all models on a dataset must either have or not have the *stream max* input. This way, all models get the same input, which makes their comparison fairer. Therefore, we applied the *stream max* input to all models running on the LOMAX dataset, and we did not apply it to models running on STEAD. For other datasets, we had to run experiments to determine whether it was better to use the *stream max* input or not. The results of these experiments are presented later in the Results section.

2.4.3. Loss functions and training procedure

The loss function used for regression tasks was mean squared error (MSE) and is shown below. Suppose the performance of the model is measured over N instances, and for each instance the model predicts the value $predicted_i$, and we denote the actual value by $ground\_truth_i$. Then the MSE can be calculated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} |predicted_i - ground\_truth_i|^2 \tag{4}$$

In the cases of classification tasks, categorical cross-entropy (CE) was used. In the case of a one-label classification where each instance i must be assigned to one of the C classes, the cross-entropy for that instance can be calculated as follows:

$$CE_i = -ln(p_i) \tag{5}$$

where p_i denotes the probability of the correct class output by the last (softmax) layer. Since training is done in batches, the single CE value is obtained by averaging the computed cross-entropies for instances within a single batch.

The goal is to minimise the loss function, whether it is a regression or a classification task. We perform this by applying the Adam optimiser with learning rate $= 0.001$, which decreases if there is no improvement in the validation loss function in four consecutive epochs. The learning rate is reduced by a factor of 0.2, but such that the learning rate is never less than $0.5 \cdot 10^{-6}$. Similarly, we add an early stop mechanism that interrupts the learning process if the loss over the validation set is not reduced within ten consecutive epochs to avoid overfitting. However, if the training process is not stopped earlier, it will be interrupted after a maximum of 250 epochs. In reviewing the papers from which we have taken the models, we found that all models converge before the 250th epoch. Therefore, this number of maximum epochs should give all models enough time to converge.

2.4.4. Validation metrics

The goal of this work was to examine how TL affects the performance and convergence of the model in specific situations. In this section, we define the metrics that should quantitatively express the performance of the model. For regression and classification tasks, we will define two metrics for performance evaluation, and the third one that shows how fast the prediction performance improves.

Classification metric.

We use a weighted F1 score to measure classification performance in classification tasks. The reason for this is the varying number of instances within individual classes in the dataset SPEECH, so a fair comparison of models based on classification accuracy is not possible. The F1 score represents the harmonic mean of the precision and recall of the test. The model has excellent precision and recall when the F1 score is equal to one, and poor when it is zero. However, this metric is not directly applicable in our cases because the classification problems in this work contain more than two classes and the F1 score is meant for binary classification problems. Therefore, we first compute the F1 score for each class separately, and in such a way that this class is compared to all other classes. This approach is known as one-vs-all F1 score. The values obtained are averaged, but in such a way that the weight of each value is proportional to the number of classes for which it is calculated (hence weighted F1 score). This is necessary to account for the uneven distribution of instances between classes.

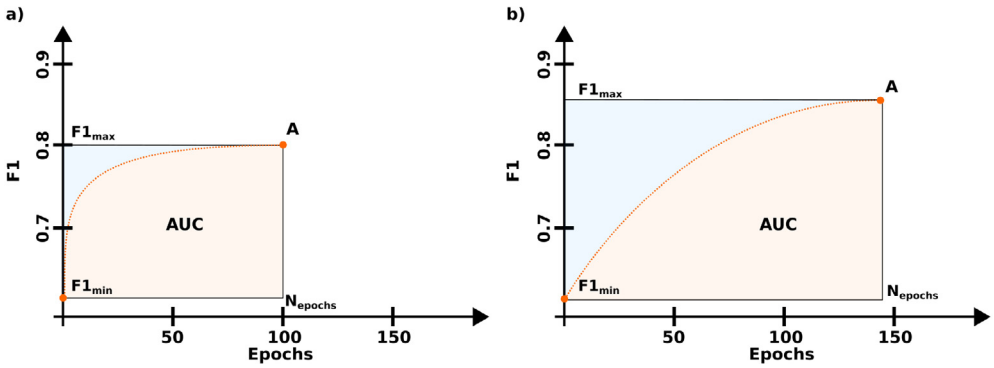

Fig. 5. An example of how the convergence rate is computed for the classification tasks.

Regression metrics.

We use the mean absolute error metric (MAE) to measure the performance of the regression models. The MAE indicates the average absolute error that the model makes. In fact, for all the regression problems presented in this paper, it does not matter whether the difference between the predicted value and the actual value is positive or negative. The unit of measure of this metric is the same as the unit of measure in the dataset to which it is applied (e.g. in the case of the S&P 500 dataset it is the dollar, and in the case of the LEN-DB dataset it is the magnitude of the earthquake) [30]. Unlike the MSE, which gives more weight to large errors and less weight to smaller ones, MAE shows an average error and therefore the percentage difference of the two models calculated between MAE is easier to interpret. MAE is calculated as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |predicted_i - observed_i| \tag{6}$$

Convergence rate.

One of the benefits of TL often mentioned in the literature is faster convergence of model optimisation due to prior knowledge. A naive way to measure this would be the number of epochs needed to complete the training. However, this metric would not be appropriate because the predictive performance of one model might increase rapidly in the first few epochs and then increase minimally over many more epochs, avoiding an early stopping mechanism, while the other model might reach its maximum performance gradually in fewer epochs and then be stopped by early stopping. Comparing these two models by epochs, we would conclude that the second model had a faster convergence due to a smaller number of epochs, while the first model actually had a much faster convergence, but it took longer for the early stopping mechanism to stop the training. Therefore, in the absence of a more appropriate measure, we introduce the convergence rate metric. We first give an example showing the intuition behind this metric, and then define it for both the classification and regression tasks.

Suppose a situation where we compare two models whose training was automatically stopped by an early stopping mechanism after the same number of epochs. Also assume that the first model improved rapidly in the first few epochs, while the second model improved uniformly over the entire training period. In this case, both models trained for the same number of epochs, but the first model has a higher convergence rate because its predictive performance improved faster.

An example of the calculation of the convergence rate for classification models is shown in Fig. 5. It can be seen that the faster the model improves, the larger the AUC area in a given rectangle. Of course, the AUC area depends on the F1 scores achieved and the number of epochs required, as can be seen in sub Figs. 5(a) and 5(b). For this reason, the AUC value cannot be used directly to compare the convergence rate of the two models. In sub Fig. 5(a), AUC is larger than in 5(b), even though the curve in 5(a) shows faster convergence. Since the convergence rate indicates how fast the model becomes good (i.e. the shape of the curve), we need to eliminate the influence of the F1 score and the number of epochs on it. This can be accomplished by observing what percentage of the area the AUC occupies in a given rectangle. If we denote the total area of a given rectangle as A and the number of epochs as N_{epochs}, then the convergence rate can be calculated as follows:

$$convergence\_rate = \frac{AUC}{A} = \frac{AUC}{(F1_{max} - F1_{min}) \cdot N_{epochs}} \tag{7}$$

The convergence rate takes the value from the range [0, 1], with higher values indicating faster convergence.

The same can be done for regression models, but unlike classification models where the tendency is to reach a higher F1 value, the goal in regression tasks is to reach a lower MAE. For this reason, the example plots for the regression task shown in Fig. 6 are inverted vertically. Another difference is that MAE can take any value in the range $[0, +\infty >$. However, this is not a problem because the convergence rate is just the ratio of the two areas. If we denote the total area of a given rectangle as A, then we can express the convergence rate for regression models as follows:

$$convergence\_rate = \frac{AUC}{A} = \frac{AUC}{(MAE_{max} - MAE_{min}) \cdot N_{epochs}} \tag{8}$$

2.4.5. Experiment workflow

The entire workflow of our experiment is shown in Fig. 7. Pretrained models are created by training all four ML model architectures on the six source datasets (three from seismology and three from other domains), resulting in a total of 24 pretrained models. It is a common practice when using TL to pretrain a model multiple times on the same dataset and then use the model with the best performance for TL. In our experiment, we follow the same approach: each pretrained model was trained from scratch (i.e. with randomly initialised weight values) ten times over the same data and the best model was selected for TL. Nevertheless, it is important to manually verify that all pretrained models have successfully converged, as this can directly affect the quality of TL. We ensure successful convergence by checking if the performance score of the model is similar to the one in the paper where the dataset used was presented.

The second step of the experiment consists of training reference models (non-TL models) from scratch and training TL models by fine-tuning the pretrained models and then comparing their performances in terms of variance in MAE and weighted F1 scores. The comparison is always performed between pairs of models (e.g. the fine-tuned pretrained MagNet model is always

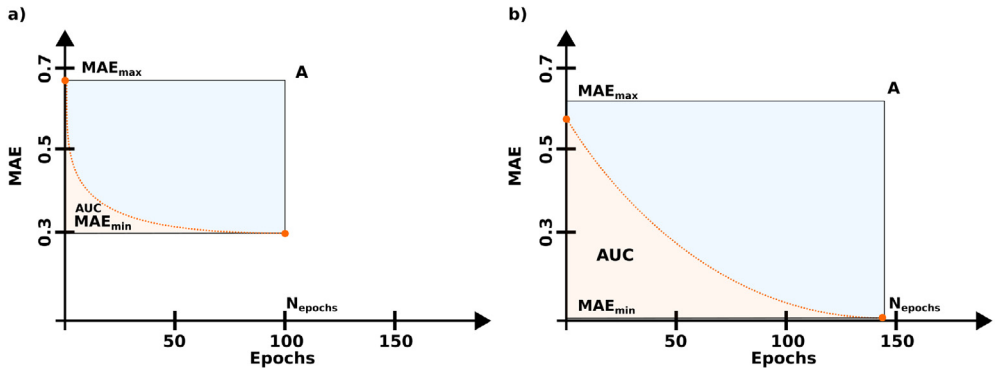

Fig. 6. An example of how the convergence rate is computed for the regression tasks.

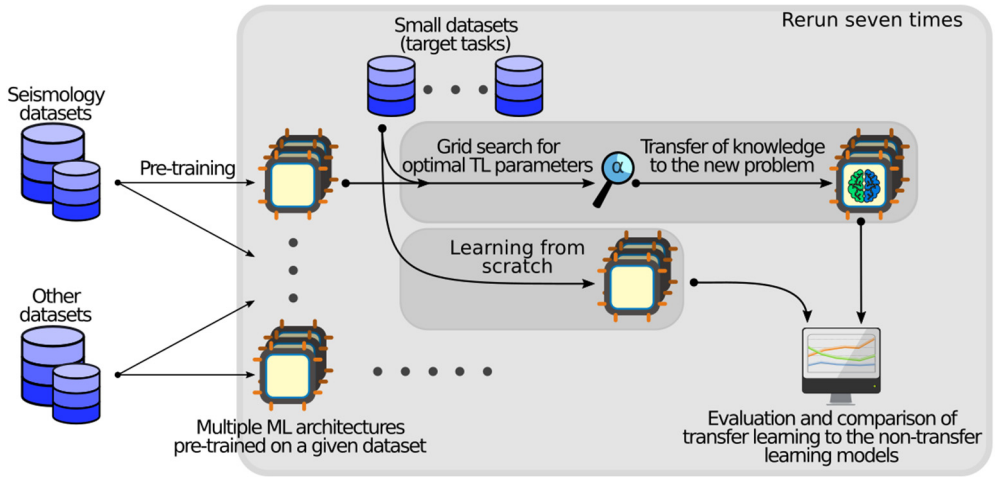

Fig. 7. Experiment workflow.

compared to the fully trained MagNet model). To confirm that the observed changes in the MAE and weighted F1 metrics are not random, we repeat the second step seven times (as shown in Fig. 7) and apply statistical tests to confirm our observations. In each run, we perform a grid search, trying ten different values for the hyperparameter (the learning rate multiplier) to find the one that gives the best results when fine-tuning the pretrained models. This step takes a lot of time and is the main reason why it was not possible to repeat the fine-tuning part more than seven times with the hardware we have.

We randomly reduced the training sets to the variants with 1500 and 9000 training instances in each run and call them *1k5* and *9k* variants, respectively. Validation and test sets remained unchanged, meaning that the models were validated and tested using the same data in each run. Therefore, no data leakage could occur between the training sets and the validation or test sets. TL models are obtained by training (fine-tuning) the pretrained models from the first step.

We should emphasise that it is not necessary to create pre-trained models from scratch in each run, as this would not affect the experiment if they were successfully trained as we have described.

2.4.6. Statistical tests

The main goal of this paper is to determine whether TL can be generally considered a useful training technique when working with TS data. For this purpose, TL models are compared with the corresponding reference models. Some of these models are regression models whose performance is measured by MAE, while others are classification models whose performance is measured by the weighted F1 score. These two metrics are not commensurable and a proper statistical test is needed for this comparison. For this reason, we count wins, losses and ties between models and apply a two-tailed sign test as proposed in [31]. This test does not assume commensurability of scores or differences, nor does it assume normal distributions. It is performed by examining each pair from the TL model and the corresponding reference model. If the TL model performs better, this is considered a win for the TL models and a loss for the reference (non-TL) models. If the TL model performs worse, it is considered a win for the reference models (non-TL models) and a loss for the TL models. It can be said that one of the two approaches (TL vs. learning from scratch) is significantly better than the other with a significance level of 0.05 if the total number of wins for the considered approach is at least $\frac{N+1.96\sqrt{N}}{2}$, where N is the total number of cases considered. If there is no difference between the two methods, then they should have approximately the same number of wins. More details on this test can be found in the referenced paper.

The second goal of this paper is to evaluate the efficiency of intra-domain and cross-domain TL. For this purpose, each pair of source and target domains is investigated. For example, a possible pair for intra-domain TL is to select the STEAD dataset as the source domain and the LEN-DB dataset (the *1k5* or the *9k* variant) as the target domain. Both datasets are from seismology, hence the intra-domain TL. Similarly, choosing the EMG dataset as the source domain and the SPEECH dataset (the *1k5* or the *9k* variant) as the target domain would be a possible example of cross-domain TL. Therefore, a separate statistical test is performed for each domain pair. The measured performance metrics are comparable since only one target domain is considered at a

time. Therefore, a metric considered will be either the MAE or the weighted F1 score, but not a mixture of both. Therefore, a statistically stronger test can be applied, namely the Wilcoxon signed-ranks test [32]. It is a nonparametric statistical test for hypothesis testing and applies to paired data (hence it is a paired difference test). The null hypothesis assumes that the mean difference between the performance metrics for TL models and reference models is zero. The alternative hypothesis to be proven in this paper is that there is a significant difference between TL models and reference models for a given target dataset. For better understanding, suppose the source dataset is EMG and the target dataset is LEN-DB (the *1k5* or the *9k* variant). This statistical test checks whether the TL model pretrained on EMG and later trained on LEN-DB performs significantly better or worse than the reference model trained from scratch on LEN-DB (the *1k5* or the *9k* variant).

Obviously, the Wilcoxon signed-ranks test is performed multiple times. This can lead to false discoveries and is a common problem in statistics, known as the multiple comparisons problem. Therefore, we apply a well-accepted two-stage Benjamini–Krieger–Yekutieli method [33] to control the false discovery rate and calculate corrected p-values. The corrected p-values are then compared to the significance level of 0.05 to either reject the null hypothesis or retain it.

2.4.7. Used software and hardware

All program code is written in the Python programming language, and was executed by a Python 3.7.7 interpreter. To train the ML model, we used the popular Tensorflow library [34] (version 1.14.0) compiled for GPUs. We used the Keras library [35], which is integrated with Tensorflow (also known as tf.keras), which made the program code simpler and easier to understand. The Python package keras-tcn (version 3.1.0) was used to instantiate the TCN model, and this package is maintained by the authors who introduced the TCN model. We used the library keras-lr-multiplier (version 0.8.0) to apply different learning rates in different layers of the models during TL. All software packages were installed within the Anaconda virtual environment. The GitHub repository contains an Anaconda environment file to automatically create a virtual environment with all the necessary packages installed to run our program and reproduce the results.

The program was run on three Dell EMC PowerEdge C4140s with two Intel Xeon Silver 4114 CPUs and 384 GB of RAM per server. Training was performed on twelve NVIDIA Tesla V100 GPUs (four per server). In the end, about 4.4 TB disk space was needed to store all data and results.

3. Results and discussion

3.1. Usefulness of stream max input

The purpose of examining the usefulness of the *stream max* input is necessary to determine to which datasets it should be applied. To this end, each model was trained twice on each dataset: in the first case, the models were trained without the *stream max* input, and in the second case, with the *stream max* input added. The results obtained in this way are compared in Table 2, where a positive value represents an increase in predictive performance. As can be seen in Table 2, the *stream max* input leads to a small improvement in the datasets LEN-DB, SPEECH and EMG, and plays a significant role for the dataset S&P 500. Therefore, we decided to use the *stream max* input for these datasets. The LOMAX and STEAD datasets were not considered in this test because their authors have indicated in their papers whether *stream max* is useful in their cases or not.

All models achieved a noticeable performance gain on the S&P 500 dataset when the *stream max* input was added. This is reasonable because it is impossible for the model to predict the absolute stock price for the next day without such an input, since the absolute size of the series is lost during data preprocessing.

3.2. General usefulness of transfer learning

The main goal of this work was to investigate whether TL can be useful for TS data in general by providing a single yes/no answer for any given experimental design. This can be achieved by counting wins and losses of TL models as explained previously in Section 2.4.6. Since the experiment was repeated seven times and also a grid search for the optimal value for the learning rate multiplier was performed each time (trying ten different multipliers), additional steps must be performed before proceeding with the statistical tests. This results in 70 trained models for each architecture–source–target triplet. Since the goal of repeating the entire experiment was to obtain more precise statistical metrics, while the goal of the grid search was to find optimal scores, a single score can be computed for each triplet by calculating the average score over all runs, considering only the best score obtained from the grid search for each run. This is done for all the triplets considered. In total, 240 values are obtained in this way. To clarify, this value is obtained by examining all possible cases for the six source domains, six target domains, two variants for each target domain, and four different ML architectures. The reader should note that the cases where the source and target datasets are the same are not considered. Repeating the whole experiment seven times also means that there are seven referent models for each target dataset. Therefore, a referent score value is calculated for each case by averaging the score values over seven repetitions of the same referent model.

It is now possible to compare the obtained TL scores with reference scores, count wins and losses, and apply a sign test. Fig. 8(a) shows the percentage of wins and losses for intra-domain TL (where the source and target domains are seismology) and (b) shows the percentage of wins and losses for cross-domain TL.

Often TL can lead to shorter training times, which can be beneficial if the ML model architectures used take a long time to train. For this reason, we look at the convergence rate of the models in Fig. 9 to get a general idea of how TL affects their convergence. It was constructed in the same way as Fig. 8, except that for each case the average convergence rate is calculated instead of the average performance score. It should be noted that the best model from the grid search is still selected based on its performance score rather than its convergence rate. This is because the primary goal is to optimise the model performance and the secondary goal is to speed up the training process. Therefore, the same models that were compared in Fig. 8 based on their performance scores are now compared in Fig. 9 based on their convergence rate.

The binomial test is applied to the data in supplementary Tables S1 and S2 which were used to generate Figs. 8 and 9. The critical value of 136 wins can be obtained using the formula for the sign test presented in Section 2.4.6, where $N = 240$. Since the TL models outperformed the reference models in 204 cases, which is shown in S1, we can conclude that TL is significantly better than training from scratch for the given experimental design.

From Fig. 8, it can be seen that intra-domain TL within seismology gives better results than cross-domain transfer to seismology (90% vs. 72%). This makes sense since the subdomains within seismology are more interconnected than with the other domains. When other domains were used as targets, seismology was as valuable as the other domains as a source domain (92% vs.

Table 2
Performance gains when using *stream max*. Positive difference in the case of LEN-DB and S&P 500 represents a decrease in MAE, while positive difference in the case of SPEECH and EMG represents an increase in the weighted F1 score.

	LEN-DB	SPEECH	EMG	S&P 500
ConvNetQuake INGV	4.79%	−0.2%	3.87%	90.44%
MagNet	1.11%	−0.3%	1.8%	96.14%
MLSTM FCN	−3.66%	1.36%	0.34%	95.17%
TCN	−0.07%	0.28%	8.29%	96.05%
Average	0.54%	0.29%	3.58%	94.45%

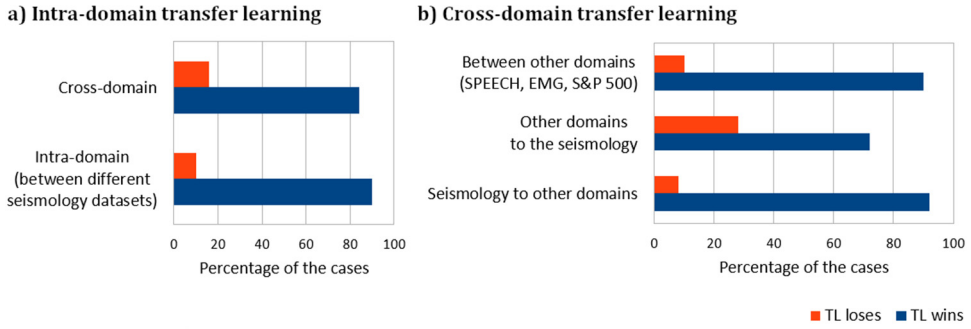

Fig. 8. Comparison of TL models and the corresponding referent models by their performance in terms of wins and losses. Subfigure (a) shows overall results for intra-domain and cross-domain scenarios, while (b) provides more insight into the cross-domain scenario. Numerical data used to generate these plots are available in the supplementary Table S1.

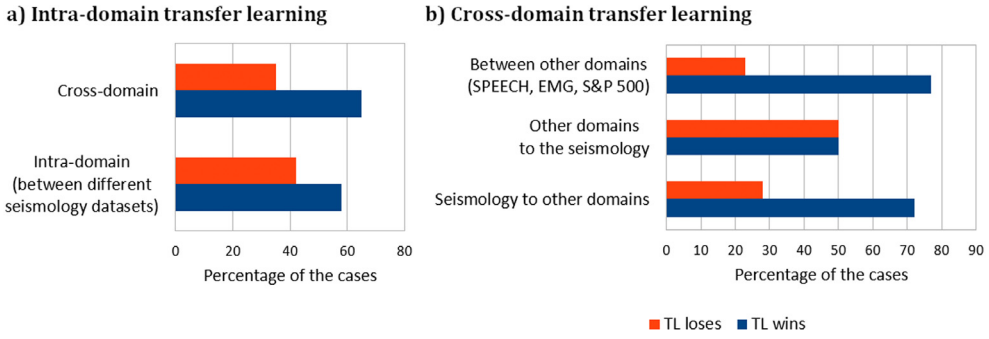

Fig. 9. Comparison of TL models and the corresponding referent models by their convergence rate in terms of wins and losses. Subfigure (a) shows overall results for intra-domain and cross-domain scenarios, while (b) provides more insight into the cross-domain scenario. Numerical data used to generate these plots are available in the supplementary Table S2.

90%). This suggests that the features learned from seismology are useful for solving problems in other domains we tested. However, TL was not as beneficial when features learned on other domains were transferred to seismology.

The asymmetry can also be seen by looking at the cases of cross-domain TL with seismology in Fig. 8(b). 92% of the TL models outperformed the reference models in the case where seismology was the source domain. In the opposite situation, where the other domains are the source and seismology is the target domain, 72% of the TL models outperformed their reference models. If there were no difference between the two groups compared, the data would show approximately the same number of wins for TL and reference models.

Looking at the comparison of convergence rates in supplementary Table S2, the data show that TL models outperformed the reference models in 153 cases, which is above the critical value of 136. We can conclude that TL enabled faster convergence in addition to better performance scores. The number of wins by convergence rate is smaller than the number of wins by performance score in all cases, leading to the general conclusion that training with TL gives better results rather than better convergence.

The correlation between performance score and convergence rate is shown in Fig. 10. The data suggest that the convergence rate does not change when TL has not obtained a better performance score. However, the convergence rate has about twice the chance ($140/77 = 1.82$) of increasing when TL achieves a better performance rating.

3.3. Domains compatibility

In the previous section we discussed the general usefulness of TL. In this section, we take a closer look at the differences in the performance and convergence rate of intra-domain and cross-domain TL. The second goal of our work was to investigate the compatibility between different pairs of domains. We hypothesise that some source domains show higher performance and convergence rate increase compared to some other source domains.

A total of 280 TL models were trained, one for each pair of source and target datasets. This number comes from the fact that four different ML architectures were tested with ten different learning rate multipliers (grid search) and the whole experiment was repeated seven times. Similarly, 28 reference models were

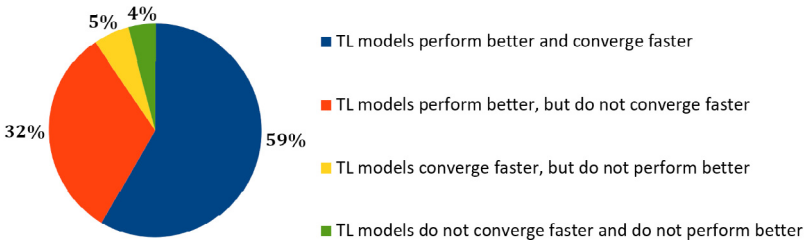

Fig. 10. The relationship between the change in convergence rate and performance score. Numerical data used to generate this plot are available in the supplementary Table S3.

Table 3
Comparison of performance scores between TL models and referent models with seismology being the target domain.

	Target datasets					
	LOMAX 1k5	LOMAX 9k	LEN-DB 1k5	LEN-DB 9k	STEAD 1k5	STEAD 9k
LOMAX	–	–	**8.21%**	**3.17%**	**6.63%**	**3.96%**
LEN-DB	**8.25%**	**4.01%**	–	–	**2.49%**	**2.52%**
STEAD	1.16%	−1.87%	0.56%	−0.52%	–	–
SPEECH	**9.29%**	**3.58%**	**4.47%**	**3.19%**	**−2.31%**	0.07%
EMG	**9.12%**	**5.34%**	−0.06%	0.41%	1.15%	**2.04%**
S&P 500	0.12%	0.12%	−0.92%	1.14%	**−7.23%**	**−2.29%**

Table 4
Comparison of performance scores between TL models and referent models when the target dataset is SPEECH, EMG or S&P 500.

	Target datasets					
	SPEECH 1k5	SPEECH 9k	EMG 1k5	EMG 9k	S&P 500 1k5	S&P 500 9k
LOMAX	**45.7%**	**871%**	**24.2%**	**17.6%**	**45.4%**	**22.8%**
LEN-DB	**309%**	**1210%**	**21.9%**	**12.1%**	**44.4%**	**22.1%**
STEAD	**323%**	**1080%**	**18.7%**	**10%**	**44.3%**	**22.7%**
SPEECH	-	-	**13.5%**	9.87%	**44.1%**	**21.6%**
EMG	**367%**	**1820%**	–	–	**41.8%**	**18.2%**
S&P 500	**122%**	**885%**	**15.2%**	**11.9%**	-	-

trained without TL. The statistical test chosen requires that the data be paired. Since the grid search was performed to find the optimal learning rate multiplier, only the model with the best performance from each replicate run is kept. This results in 28 TL models coming from four different ML architectures and seven reruns. For each source–target domain pair, a difference in performance scores between TL models and reference models is calculated. Then, a single difference value is obtained by averaging the calculated differences.

Tables 3 and 4 are created according to the described procedure. Table 3 shows situations where the target domain is seismology, while Table 4 shows situations where the target dataset is from the other selected domains. The difference between the performances is shown as a percentage for easier comparison. A positive percentage difference represents a decrease in MAE for regression models and an increase in the weighted F1 score for classification. A negative percentage difference represents an increase in MAE for regression models and a decrease in the weighted F1 score for classification models. Thus, a positive difference always represents a positive impact of TL, while a negative difference represents a negative impact of TL.

A separate Wilcoxon signed-ranks test is performed for each pair of domains, resulting in a total of 60 statistical tests (30 in each table). Therefore, the significance level of 0.05 requires adjustment for this problem. To this end, we use the method Benjamini–Krieger–Yekutieli, to combat the multiple comparisons problem and adjust the significance level accordingly. Coloured cells are found to be statistically significant. The green colour shows that the TL models perform better than the reference models (positive transfer), while the red colour shows the opposite situation (negative transfer).

The data in Table 3 show that it was beneficial to transfer the features learned on the LOMAX and LEN-DB datasets to other seismological datasets. The same cannot be said for the STEAD dataset, as the models pretrained on this dataset did not show statistically significant differences compared to the models trained from scratch when fine-tuned on the other two seismological datasets. Since the LEN-DB and STEAD datasets contain local earthquakes, the difference being that the instrument response is removed from the LEN-DB while it is not removed from the STEAD waveforms, and the sampling rates of the data are 20 Hz and 100 Hz for LEN-DB and STEAD, respectively, we speculate that this may account for why the transfer of knowledge from LEN-DB to STEAD is useful while it is not useful in the other direction. The LOMAX dataset, as a dataset of earthquakes recorded at any distance, also has the instrument response removed from the waveforms and the data are sampled at a sampling rate of 20 Hz, which could explain the successful transfer between LOMAX and LEN-DB in both directions and a one-way transfer from LOMAX to STEAD. Another important difference between STEAD and the other two seismological datasets is that the *stream max* input was not used in the pretraining and training of the models on STEAD.

Knowledge transfer from non-seismological domains to STEAD and LEN-DB datasets does not have the same impact on the results despite their similarities. Models pretrained on EMG had no statistically significant improvement for LEN-DB, while they were useful for STEAD 9k. However, models pretrained on SPEECH were an adequate choice for LEN-DB, while they were a poor choice for STEAD 1.5k due to negative transfer.

From Tables 3 and 4, it can be seen that negative transfer was observed only when the source domain was either SPEECH or S&P 500 and the target domain was STEAD. In the case of SPEECH

as the source domain, we found that negative transfer occurred in all models except MagNet. In the case of the S&P 500 source domain, the same was observed for the *1k5* variant, while for the *9k* variant MLSTM FCN was also no longer affected along with the MagNet. Both examples show how the effects of negative transfer decrease (approach zero) as the size of the target training dataset becomes larger.

Looking at the case of SPEECH as the target dataset in Table 4, we can see an unusually large increase in performance. For this reason, we examined this case in more detail. We found that all referent models except the MLSTM FCN did not converge on both *1k5* and *9k* variants. TL proved beneficial in these cases for all models regardless of source domains. This is true even for the models pretrained on the S&P 500 dataset, which can largely be considered "random" and almost "impossible to predict". The results suggest that the SPEECH dataset is very challenging due to a small number of training instances and TL is of great help in this case.

In the case of the EMG target dataset, the table shows that the models pretrained on SPEECH were a good choice for the *1k5* variant, while in the case of the *9k* variant, these models achieved the same results as the reference models. In contrast, the models pretrained on other datasets performed better than the reference models for both the *1k5* and *9k* variants. This suggests that some source domains are useful even when a larger set of training instances is available, while some other domains are not. Besides SPEECH, this was the only target domain that benefited from *a priori* knowledge gained in the S&P 500 dataset.

In the case of S&P 500 as the target dataset, it seems that the TL boost remains almost the same regardless of the source dataset. We investigated this case in more detail. We found that MAE of the ConvNetQuake INGV and MLSTM FCN models was about five times higher than that of the other models. The same problem existed only for the ConvNetQuake INGV in the case of the *9k* variant. This is the reason why the boost for the *9k* variant was halved compared to the *1k5* variant. TL helped these two models get about the same MAE as all other models.

It is logical to expect that the effects of TL become less noticeable as the size of the target training set increases. The reason is that as the size of the target dataset increases, more and more training instances are available, leading to better performance score. This automatically reduces the need for TL, since the target dataset alone contains enough data for successful training. This seems to be true in most cases. For example, in the case where the source dataset is EMG and the target domain is LOMAX, the performance gain is lower for the *9k* variant than for the *1k5* variant. However, there are a few cases where the models on the *9k* variant achieve a larger performance gain compared to the *1k5* variant. These cases are: EMG → STEAD and LEN-DB → STEAD.

We studied these cases in more detail. We hypothesise that this may be because the STEAD waveforms do not have the instrument response removed. In addition, the input size of STEAD is significantly larger (6000 sampling points per channel) than the other local earthquakes dataset LEN-DB (540 sampling points per channel), which affects the number of model parameters. Both of these things make the learning process more difficult. This would suggest that the usefulness of TL depends on the size of the target training set. When the target training set is extremely small, referent and TL models perform equally well and there is no benefit of TL because there are not enough training instances to learn a meaningful representation. As the size of the target training set increases, the difference in performance scores between TL models and reference models increases. At some point, the target training set contains enough data for successful training from scratch. From this point on, the benefits of TL begin to disappear as the size of the target training set continues to grow. The same is true in the case of negative transfer, when the models perform worse than the reference models. As explained earlier, these negative effects also diminish as the size of the target training set increases. Of course, these "boundary" sizes are different for each domain (i.e. depending on the complexity of the task) and the ML architecture used, which may explain why reference and TL models perform equally well on some target datasets, while they perform differently on other domains for the same target dataset size.

Tables 5 and 6 show the change in convergence rate for each pair of TL and reference models examined in Tables 5 and 6. A difference in their convergence rate is calculated and these differences are averaged and expressed as a percentage with respect to the average convergence rate of the reference models. In this way, a single value is obtained for each cell in the table. The statistical tests and Benjamini–Krieger–Yekutieli correction are performed in the same way as for Tables 5 and 6.

Table 5 shows that the knowledge gained on the LOMAX and LEN-DB datasets did not statistically significantly accelerate the convergence rate on the STEAD target tasks. On the other hand, the knowledge gained on STEAD accelerated the convergence on the LOMAX and LEN-DB target tasks. Moreover, the table shows that the fine-tuned models on LOMAX and LEN-DB achieved faster convergence in more cases due to the knowledge transfer than on STEAD. This is in contrast to the observations made by the performance score for the same cases. This is an example that shows that an increase in performance does not necessarily lead to an increase in convergence rate, as inferred in Section 3.2.

The data suggest that almost all models show no statistically significant differences in convergence compared to referent models on the SPEECH target tasks. However, this comparison is not as meaningful because we previously found that these referent models did not learn a meaningful representation (Table 3).

EMG target datasets are the only ones where we observed significantly slower convergence of the TL models compared to the reference models on the larger dataset, while the convergence rate was significantly faster on the smaller one. Upon closer inspection of this case, we found that all four architectures exhibited this behaviour. The reason for this behaviour is unclear.

In the case of the S&P 500 target dataset, we found that the convergence of the ConvNetQuake INGV, MagNet, and MLSTM FCN models was accelerated by the TL. For each model, this speedup was approximately the same across all source domains. This explains why it appears that the speedup is constant regardless of the source domain.

For more detailed results, the reader is referred to the supplementary Tables S5 and S6.

3.4. The influence of models on TL

Fig. 11 shows how often each TL model outperformed the corresponding reference model. It is important to note that this plot does not reflect performance scores, but the extent to which *a priori* knowledge was used by the models.

Fig. 11 shows that TL results can vary significantly depending on the model chosen. For example, MagNet and MLSTM FCN were found to make the best use of pre-learned knowledge in intra-domain scenarios, while ConvNetQuake INGV and TCN performed significantly worse. However, the success of MagNet, MLSTM FCN and TCN in using TL in cross-domain transfer to seismology dropped significantly. The reason for the decrease is the lower compatibility of knowledge in transferring from other domains to seismology, as identified earlier. Meanwhile, ConvNetQuake INGV performed almost as well as in the intra-domain scenario. Even though ConvNetQuake INGV and MagNet were both primarily developed for the domain of seismology, they show different

Table 5
Comparison of convergence rate between TL models and referent models with seismology being the target domain.

	Target datasets					
	LOMAX 1k5	LOMAX 9k	LEN-DB 1k5	LEN-DB 9k	STEAD 1k5	STEAD 9k
LOMAX	–	–	9.08%	−3.15%	16.6%	11.6%
LEN-DB	**11.6%**	6.72%	–	–	17.5%	11.8%
STEAD	**16.5%**	4.59%	**22.6%**	7.59%	–	–
SPEECH	10.2%	3.19%	**5.45%**	1.35%	6.73%	8.71%
EMG	10.1%	5.76%	14.9%	−0.31%	8.98%	12.8%
S&P 500	4.04%	4.33%	13.8%	−1.39%	**9.76%**	13.9%

Table 6
Comparison of convergence rate between TL models and referent models when the target dataset is SPEECH, EMG or S&P 500.

	Target datasets					
	SPEECH 1k5	SPEECH 9k	EMG 1k5	EMG 9k	S&P 500 1k5	S&P 500 9k
LOMAX	0.12%	4.55%	13.3%	−1.89%	6.37%	5.39%
LEN-DB	1.13%	7.87%	17.0%	−3.67%	6.43%	3.66%
STEAD	2.35%	3.4%	17.3%	-7.19%	6.16%	9.7%
SPEECH	-	-	13.9%	−5.11%	6.13%	3.13%
EMG	5.76%	9.35%	–	–	6.2%	8.25%
S&P 500	4.8%	5.75%	8.12%	-6.09%	-	-

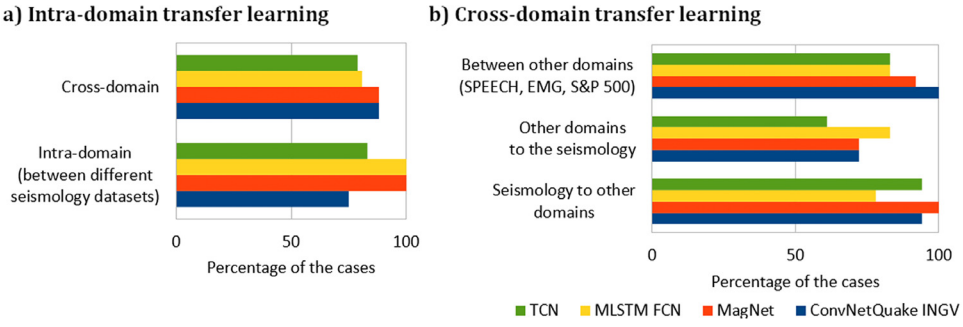

Fig. 11. The influence of the chosen model on the TL performance gain. The plots depict how many times each TL model outperformed its corresponding referent model. Subfigure (a) shows overall results for intra-domain and cross-domain scenarios, while (b) provides more insight into the cross-domain scenario. Numerical data used to generate these plots are available in the supplementary Table S4.

behaviours in intra-domain and cross-domain knowledge transfer to seismology. The same can be said for the general purpose models: MLSTM FCN proved to be the better option when the target domain was seismology, while they performed equally well or TCN better—in the other two cases.

Overall, all models used *a priori* knowledge in 80% — −90% of the cases studied. Therefore, one might expect to obtain some of the benefits of TL with a preselected model, but must be aware of the fact that better performance may be obtained by a different model.

The extent to which the *a priori* knowledge is useful also depends on the hyperparameters of the TL. In our case, we explored ten different values for the learning rate multiplier hyperparameter – by grid search – and present the results in Fig. 12. The figure illustrates how different learning rate multipliers affect the performance of the models when trained with TL, expressed as a percentage. A positive percentage means that TL outperformed the reference models and vice versa. It can be seen that different multiplier values have noticeably different effects on the extent to which the TL approach will be successful. ConvNetQuake INGV makes best use of the *a priori* knowledge when the multiplier value is small, and the best performance was achieved at a value of 0.25. TCN is similar to ConvNetQuake INGV in the sense that both models consist of a larger number of convolutional layers, but regardless of this similarity, TCN was found to perform best at multipliers of 0.75 and above.

MagNet and MLSTM FCN models are similar in the sense that both have fewer convolutional layers and contain LSTM units. Notwithstanding these similarities, they also exhibit different behaviour. MLSTM FCN works best with multiplier values of 0.5 or higher. For the MagNet model, the greatest gain was found at a multiplier of 1.0 or 0.01. It experiences effects of negative transfer with multipliers greater than 0.1, with 1.0 being the exception. This exception clearly distinguishes the behaviour of the MagNet model from that of the other models studied.

Practitioners mostly use learning rate multipliers smaller than 1.0 as an indirect mechanism to retain knowledge from the source task and use it to maximise performance results on the target task. However, Fig. 12 shows that multipliers greater than 1.0 can also be beneficial in some models. This suggests that even with learning rate multipliers greater than 1.0, some models can take advantage of the *a priori* knowledge gained and that the knowledge is not "lost" in convolutional layers.

4. Conclusion

In this paper, we have comprehensively studied the impact of TL on TS data to find out how successful TL can be in solving the upcoming problems in implementing deep learning models for TS

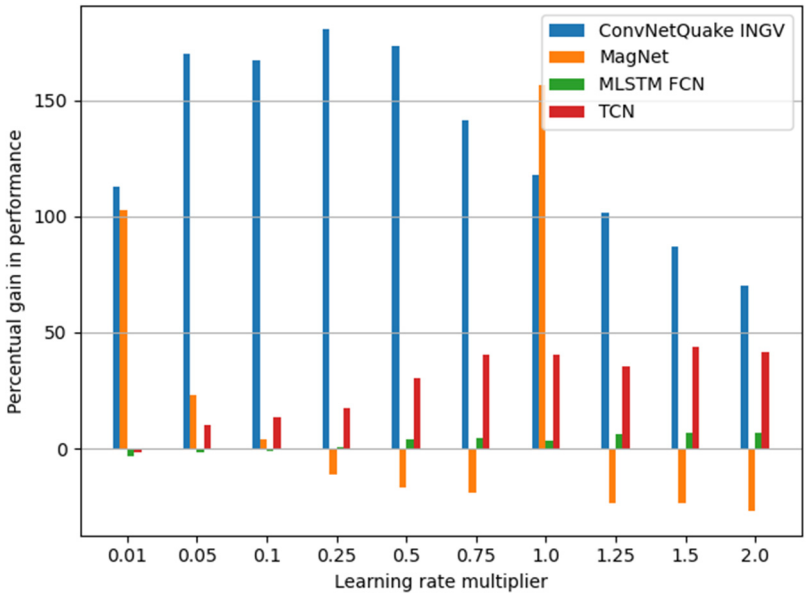

Fig. 12. Influence of learning rate multiplier on TL performance gain. The performance gain is calculated separately for each model and learning rate multiplier. It represents the average increase in predictive performance of the TL models compared to the reference models over all pairs of source and target datasets.

data. As models become deeper and more complex, the need for larger training sets increases. TL has been well studied in some areas (e.g. image classification) and has proven to be one of the good ways to solve the problem of smaller training sets. Since we focused only on TS data, the conclusions of our research can only be applied to TS data, and extending it to other types of data requires further research.

We conducted an experiment in which we investigated both intra-domain TL situations between different seismological datasets and cross-domain TL between seismology, sound, medical and financial datasets. We ensured that all selected datasets had sufficient size for successful pretraining, and we performed a grid search to find the optimal hyperparameters for successful knowledge transfer. Finally, we observed the effects of TL on model performance and convergence rate by comparing TL models to models trained from scratch. Models were compared to two different sized training datasets for the target domain, one containing 1500 training instances and the other containing 9000 training instances.

Our experiment showed that there was a statistically significant difference in performance and convergence rate between TL models and models trained from scratch. There was an improvement in model performance 76% of the time, while there was an acceleration in convergence 64% of the time. The data showed that if knowledge transfer from the first to the second domain leads to improvement, it is not necessary that knowledge transfer in the opposite direction also leads to improvement. TL models were twice as likely to converge faster than models trained from scratch when TL led to better performance.

When we reached high-level conclusions, we proceeded to examine each pair of source–target domains separately. The SPEECH target dataset proved to be very challenging with small training sets. The vast majority of models trained from scratch failed to learn anything on such small datasets, while TL models managed to converge. This is true even for the transfer of knowledge acquired on the S&P 500 dataset, which can be considered "random" and "unpredictable". In other cross-domain cases, the TL models far outperformed the models trained from scratch. In intra-domain TL cases, knowledge has been shown to be transferable between different seismological datasets, but we also observed that other things could be a factor on which the success of knowledge transfer depends.

Finally, we have investigated the influence of model selection on the success of knowledge transfer. Our data show that, in general, all four models outperformed the models trained from scratch 80–90% of the time. However, in some cases, not all models were equally successful. The success of the knowledge transfer is strongly influenced by the hyperparameters related to the TL, and in our case it was only the learning rate multiplier that adjusts the fine-tuning process of the convolutional layers. We found the optimal value of this hyperparameter for each model in each scenario by grid search; it was not the same for all models, but in most cases it was less than or equal to 1.0. However, in some cases and for some models, values greater than 1.0 were shown to work well, which could be explored in terms of differential learning rate research in future work.

We started from the idea that all TS data are essentially signals, that all TS data can be decomposed into a linear combination of sine and cosine waves, which may indicate that there is shared knowledge that can be used to solve problems in different TS domains. We experimented with six intra-domain and 24 cross-domain scenarios to determine the extent to which this assumption holds. We found only two source–target domain pairs where the TL models performed statistically significantly worse than the reference models (i.e. negative transfer). This suggests that when TL is applied without prior knowledge of source–target domain compatibility, one is very likely to obtain better or at least equally good performance as the model trained from scratch. By analysing pairs of source and target domains, we were able to determine how even seemingly unrelated domains can be compatible enough to yield positive effects (i.e. positive transfer). Ultimately, we found that all models had about the same probability of producing better outcomes with knowledge transfer. All this leads to the conclusion that pretraining models in TS is useful and should be given a chance, even if the two TS tasks seem unrelated. The application of TL to TS has yet to become popular and powerful, as it has already *revolutionised* many other domains.

Abbreviations

AUC	area under the curve
CNN	convolutional neural network
EMG	electromyography
GPU	graphical processing unit
LSTM	long short-term memory
MAE	mean absolute error
ML	machine learning
MLSTM FCN	multivariate LSTM fully convolutional network
MSE	mean squared error
RNN	recurrent neural network
TCN	temporal convolutional network
TL	transfer learning
TS	time series

CRediT authorship contribution statement

Erik Otović: Conceptualisation, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualisation, Writing - original draft, Writing – review & editing. **Marko Njirjak:** Methodology, Data curation, Validation, Software, Writing – original draft. **Dario Jozinović:** Conceptualisation, Data curation, Resources, Investigation, Writing – original draft, Writing - review & editing. **Goran Mauša:** Funding acquisition, Resources, Writing – review & editing. **Alberto Michelini:** Funding acquisition, Resources, Writing – review & editing. **Ivan Štajduhar:** Conceptualisation, Funding acquisition, Methodology, Project administration, Resources, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was supported in part by the Croatian Science Foundation projects UIP-2019-04-7999, DOK-2020-01-4659 and IP-2020-02-3770, and the COST project G2Net CA17137 A network for Gravitational Waves, Geophysics and Machine Learning. This work was also partially supported by the project INGV Pianeta Dinamico 2021 Tema 8 SOME (CUP D53J1900017001) funded by Italian Ministry of University and Research "Fondo finalizzato al rilancio degli investimenti delle amministrazioni centrali dello Stato e allo sviluppo del Paese, legge 145/2018".

This research was conducted using the resources of the Isabella Computer Cluster located in SRCE - University of Zagreb University Computing Centre.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.knosys.2021.107976.

References

[1] L. Shao, F. Zhu, X. Li, Transfer learning for visual categorization: A survey, IEEE Trans. Neural Netw. Learn. Syst. 26 (5) (2015) 1019–1034, http://dx.doi.org/10.1109/TNNLS.2014.2330900.

[2] Y.-P. Lin, T.-P. Jung, Improving EEG-based emotion classification using conditional transfer learning, Front. Hum. Neurosci. 11 (2017) 334.

[3] T. Wen, R. Keyes, Time series anomaly detection using convolutional neural networks and transfer learning, 2019, arXiv preprint arXiv:1905.13628.

[4] Q. Hu, R. Zhang, Y. Zhou, Transfer learning for short-term wind speed prediction with deep neural networks, Renew. Energy 85 (2016) 83–95, http://dx.doi.org/10.1016/j.renene.2015.06.034.

[5] D. Jozinović, A. Lomax, I. Štajduhar, A. Michelini, Transfer learning: Improving neural network based prediction of earthquake ground shaking for an area with insufficient training data, Geophysical Journal International (2021) http://dx.doi.org/10.1093/gji/ggab488.

[6] K. Kashiparekh, J. Narwariya, P. Malhotra, L. Vig, G. Shroff, Convtimenet: A pre-trained deep convolutional neural network for time series classification, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE, 2019, pp. 1–8.

[7] R. Ye, Q. Dai, A novel transfer learning framework for time series forecasting, Knowl.-Based Syst. 156 (2018) 74–99.

[8] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Transfer learning for time series classification, in: 2018 IEEE International Conference on Big Data (Big Data), 2018, pp. 1367–1376, http://dx.doi.org/10.1109/BigData.2018.8621990.

[9] A. Lomax, D. Jozinović, E. Otović, A. Michelini, Seismogram waveform datasets for ConvNetQuake_Ingv, 2021, http://dx.doi.org/10.5281/zenodo.5040865, This research was supported by the EC ARISTOTLE Project ECHO/SER/2015/722144. URL https://doi.org/10.5281/zenodo.5040865.

[10] A. Lomax, A. Michelini, D. Jozinović, An investigation of rapid earthquake characterization using single-station waveforms and a convolutional neural network, Seismol. Res. Lett. 90 (2A) (2019) 517–529, http://dx.doi.org/10.1785/0220180311, arXiv:https://pubs.geoscienceworld.org/srl/article-pdf/90/2A/517/4655400/srl-2018311.1.pdf.

[11] F. Magrini, D. Jozinović, F. Cammarano, A. Michelini, L. Boschi, Local earthquakes detection: A benchmark dataset of 3-component seismograms built on a global scale, Artif. Intell. Geosci. 1 (2020) 1–10.

[12] F. Magrini, D. Jozinović, F. Cammarano, A. Michelini, L. Boschi, LEN-DB - Local earthquakes detection: a benchmark dataset of 3-component seismograms built on a global scale, 2020, http://dx.doi.org/10.5281/ZENODO.3648231.

[13] S.M. Mousavi, Y. Sheng, W. Zhu, G.C. Beroza, Stanford earthquake dataset (STEAD): A global data set of seismic signals for AI, IEEE Access 7 (2019) 179464–179476, http://dx.doi.org/10.1109/ACCESS.2019.2947848.

[14] P. Warden, Speech commands: A dataset for limited-vocabulary speech recognition, 2018, arXiv:1804.03209.

[15] E. Donati, EMG From forearm datasets for hand gestures recognition, 2019, http://dx.doi.org/10.5281/zenodo.3194792.

[16] E. Donati, M. Payvand, N. Risi, R. Krause, K. Burelo, G. Indiveri, T. Dalgaty, E. Vianello, Processing EMG signals using reservoir computing on an event-based neuromorphic system, in: 2018 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2018, pp. 1–4.

[17] S. Lobov, N. Krilova, I. Kastalskiy, V. Kazantsev, V. Makarov, Latent factors limiting the performance of sEMG-interfaces, Sensors 18 (4) (2018) 1122, http://dx.doi.org/10.3390/s18041122.

[18] S. Benatti, F. Casamassima, B. Milosevic, E. Farella, P. Schönle, S. Fateh, T. Burger, Q. Huang, L. Benini, A versatile embedded platform for EMG acquisition and gesture recognition, IEEE Trans. Biomed. Circuits Syst. 9 (5) (2015) 620–630.

[19] J. Eapen, D. Bein, A. Verma, Novel deep learning model with CNN and bi-directional LSTM for improved stock market index prediction, in: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), 2019, pp. 0264–0270.

[20] J. Chou, T. Nguyen, Forward forecast of stock price using sliding-window metaheuristic-optimized machine-learning regression, IEEE Trans. Ind. Inf. 14 (7) (2018) 3132–3142.

[21] T. Perol, M. Gharbi, M. Denolle, Convolutional neural network for earthquake detection and location, Sci. Adv. 4 (2) (2018) e1700578.

[22] S.M. Mousavi, G.C. Beroza, A machine-learning approach for earthquake magnitude estimation, Geophys. Res. Lett. 47 (1) (2020) http://dx.doi.org/10.1029/2019GL085976, e2019GL085976 e2019GL085976 2019GL085976. arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2019GL085976.

[23] F. Karim, S. Majumdar, H. Darabi, S. Harford, Multivariate LSTM-FCNs for time series classification, Neural Netw. 116 (2019) 237–245.

[24] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, 2018, arXiv:1803.01271.

[25] S. Bozinovski, Reminder of the first paper on transfer learning in neural networks, 1976, Informatica 44 (3) (2020).

[26] Q. Yang, S. Pan, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2010) 1345–1359, http://dx.doi.org/10.1109/TKDE.2009.191.

[27] N. Tajbakhsh, J.Y. Shin, S.R. Gurudu, R.T. Hurst, C.B. Kendall, M.B. Gotway, J. Liang, Convolutional neural networks for medical image analysis: Full training or fine tuning? IEEE Trans. Med. Imaging 35 (5) (2016) 1299–1312, http://dx.doi.org/10.1109/tmi.2016.2535302.

[28] N.M. Nawi, W.H. Atomi, M. Rehman, The effect of data pre-processing on optimized training of artificial neural networks, Proc. Technol. 11 (2013) 32–39, http://dx.doi.org/10.1016/j.protcy.2013.12.159, 4th International Conference on Electrical Engineering and Informatics, ICEEI 2013.
[29] S.M. Mousavi, G.C. Beroza, A machine-learning approach for earthquake magnitude estimation, Geophys. Res. Lett. 47 (1) (2020) e2019GL085976, http://dx.doi.org/10.1029/2019GL085976, e2019GL085976 2019GL085976.
[30] A. Botchkarev, Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology, 2018, arXiv:1809.03006.
[31] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[32] W. F., Individual comparisons by ranking methods, Biom. Bull. 1 (6) (1945) 80–83.
[33] Y. Benjamini, A.M. Krieger, D. Yekutieli, Adaptive linear step-up procedures that control the false discovery rate, Biometrika 93 (3) (2006) 491–507, http://dx.doi.org/10.1093/biomet/93.3.491, arXiv:https://academic.oup.com/biomet/article-pdf/93/3/491/1080958/933491.pdf.
[34] Tensorflow, http://dx.doi.org/10.5281/zenodo.4758419, URL "https://github.com/tensorflow/tensorflow/graphs/contributors".
[35] F.c. Chollet, et al., Keras, 2015, https://keras.io.