Finding Foundation Models for Time Series Classification with a PreText Task

Ali Ismail-Fawaz[1], Maxime Devanne[1], Stefano Berretti[2], Jonathan Weber[1], and Germain Forestier[1,3]

[1] IRIMAS, Universite de Haute-Alsace, Mulhouse France
{ali-el-hadi.ismail-fawaz,maxime.devanne,jonathan.weber,germain.forestier}@uha.fr
[2] MICC, University of Florence, Florence Italy stefano.berretti@unifi.it
[3] DSAI, Monash University, Melbourne Australia germain.forestier@monash.edu

Abstract. Over the past decade, Time Series Classification (TSC) has gained an increasing attention. While various methods were explored, deep learning – particularly through Convolutional Neural Networks (CNNs) –stands out as an effective approach. However, due to the limited availability of training data, defining a foundation model for TSC that overcomes the overfitting problem is still a challenging task. The UCR archive, encompassing a wide spectrum of datasets ranging from motion recognition to ECG-based heart disease detection, serves as a prime example for exploring this issue in diverse TSC scenarios. In this paper, we address the overfitting challenge by introducing pre-trained domain foundation models. A key aspect of our methodology is a novel pretext task that spans multiple datasets. This task is designed to identify the originating dataset of each time series sample, with the goal of creating flexible convolution filters that can be applied across different datasets. The research process consists of two phases: a pre-training phase where the model acquires general features through the pretext task, and a subsequent fine-tuning phase for specific dataset classifications. Our extensive experiments on the UCR archive demonstrate that this pre-training strategy significantly outperforms the conventional training approach without pre-training. This strategy effectively reduces overfitting in small datasets and provides an efficient route for adapting these models to new datasets, thus advancing the capabilities of deep learning in TSC.

Keywords: Time Series Classification · Deep Learning · Pre-Training Deep Learning · Time Series · Convolutional Neural Networks.

1 Introduction

Time series are sequences of data points indexed by time, typically obtained by observing a random variable over consistent intervals. These data sequences are prevalent in various machine learning applications, including classification [14] and clustering [8], among others. Over the past decade, Time Series Classification

This is the author's version of an article published at PAKDD Workshop IWTA 2024. The final authenticated version is available online at: https://doi.org/10.1007/978-981-97-2650-9_10

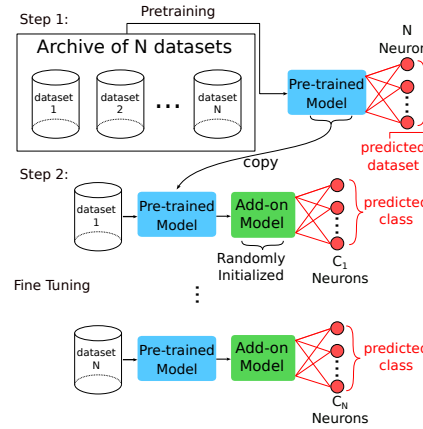

Fig. 1. Summary of the proposed pretext task approach. Given an archive of N datasets, the first step is to train a *pre-trained* model (in blue) on all of the datasets, where the classification task is to predict the dataset each time series belongs to. The second step is to copy the *pre-trained* model and follow it with an *addon* model (in green) randomly initialized. The second step is done independently for each of the N datasets of the archive. After constructing the N new models, they are fine-tuned on each dataset depending on the task of each one.

(TSC) has witnessed a surge in research activity. This increasing interest spans across diverse fields such as medicine and telecommunications.

Deep learning, with its advanced neural network architectures, offers significant potential for TSC classification [11], often achieving state-of-the-art performance in various TSC tasks. Conventionally, solving a TSC problem with deep learning involves initializing a neural network architecture randomly and feeding it with the training data. However, when the training dataset is limited, this method can lead to overfitting, where the model adapts too closely to the training data, resulting in poor performance on unseen test samples. This challenging problem of having a dataset with few training examples does exist almost everywhere in machine learning research. This common problem reflects a real case scenario and it has been adapted to datasets of the UCR archive, the most comprehensive repository for univariate TSC datasets. This large archive is composed of 128 datasets covering various TSC tasks going from motion recognition to the classification of heart diseases using Electrocardiogram (ECG) signals. The depth of the UCR archive lies in its diverse representation of tasks across multiple domains, often providing several example datasets for each domain.

Gathering additional training samples to address the overfitting issue can be time-consuming and resource-intensive. Furthermore, even if more samples are generated, annotating them typically necessitates expertise, thus introducing additional costs. As a solution, various approaches were proposed in the literature such as data augmentation [9], and the use of hand-crafted generic filters [6]. However, while effective, these methods can introduce noise and disrupt the training process.

To take advantage of having multiple datasets within a given domain, we aim to identify a foundation pre-trained model for each domain of TSC, replacing the random initialization used in traditional techniques. This *pre-trained* foundation model is trained on a shared task among the different datasets. Specifically, the task is to predict the original dataset of each sample. For instance, if we merge two datasets, *dataset1* and *dataset2*, from the same domain, and temporarily

disregard their specific target classes, the objective of the pre-trained model becomes discerning the origin of each sample in this combined set.

Once the pre-training phase is completed, the model is fine-tuned for the specific tasks of each dataset. An overview of our proposed methodology is depicted in Figure 1. After the pre-trained model has been fully trained on the pretext task, the fine tuning stage can follow two different options. The first option is to fine tune the pre-trained model followed by a classification layer with respect to the classification task of the dataset. The second option is to fine tune the pre-trained model cascaded with deeper layers to extract deeper features followed by a classification layer. The first option was followed in the work of [10], where the authors studied the effect of transfer learning on TSC. However, performance was not as good as expected, due to the fact that most target datasets were sensitive on the dataset used as source for the transfer learning.

In this work, we follow the setup of the second option. In particular, we believe that in the first option ignoring deeper meaningful features correlated with one dataset during the fine tuning step implies a strong assumption: the pre-trained model learned the optimal convolution filters that are able to correctly generalize to the classification task. But this may not be the case.

In summary, the main contributions of this work are:

- Novel domain foundation models trained to solve a pretext task to enhance deep learning for TSC;
- Novel Batch Normalization Multiplexer (BNM) layer that controls the multi-dataset (multi-distribution) problem of the batch normalization;
- Extensive experiments on the UCR archive show a significant improvement when using the pre-trained model over the baseline model.

2 Related Work

Many works in the literature have been proposed to address the TSC task and have been evaluated on the UCR archive. These tasks range from similarity based approaches to ensemble models, deep learning, *etc.* In what follows, we present the latest state-of-the-art approaches that addressed the TSC task.

2.1 Deep Learning Techniques

In 2019, the authors of [11] released a detailed review on the latest deep learning approaches for solving TSC on the UCR archive. The two best performing models were Convolutional Neural Networks (CNNs), the Fully Convolutional Network (FCN), and the Residual Network (ResNet) [15]. Moreover, the authors of [12] proposed a new CNN based architecture called InceptionTime, which is an ensemble of multiple Inception models. More recently, new hand-crafted convolution filters were proposed to enhance InceptionTime by [6] with their proposed H-InceptionTime model. It achieves new state-of-the-art performance for deep learners on TSC. Finally, the authors of [4] argued that there is no need for large

complex models to solve the TSC task on the UCR archive, but instead they proposed a lighter architecture called LITE. LITE balances between its small number of parameters and its state-of-the-art performance using some boosting techniques.

2.2 Pre-Training Deep Learning Techniques

In the last few years, some approaches addressed the TSC task using pre-trained deep learning models. For instance, the work in [10] proposed to apply transfer learning of a deep learning model from a source time series dataset to a target dataset. The deep learning model was trained on a source dataset and then fine tuned on a target dataset. Some works trained a deep learning model with a Self-Supervised task and then used its output features to learn a classifier [7]. The so called "knowledge distillation" is another technique that uses pre-trained models. Following such idea, the authors of [1] used a pre-trained FCN [15] model and distilled its knowledge to a smaller version of FCN. This process helps to balance between a smaller architecture and its performance.

The difference between our proposed approach and the traditional pre-training techniques is the usage of multiple domains during training. It is important to note that the goal of this work is not to solve transfer learning but instead to enhance deep learners when solving direct TSC tasks using a pre-training approach. In what follows, we detail our approach and the used pretext task.

3 Proposed Method

3.1 Pretext Task

A Univariate Time Series (UTS) $\mathbf{x} = \{x_0, x_1, \ldots, x_T\}$ is a vector of T values of a random variable changing with time. Univariate Time Series Classification Dataset (UTSCD) $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N-1}$ is a set of N UTS with their corresponding label vector $\mathbf{y}$. We denote by C the number of unique labels existing in D. Given a backbone deep learning model for TSC made of n layers, we divided the backbone model into two sub-models. The first sub-model (referred to as the pre-trained model) focuses on learning a pretext task; the latter is an additional randomly initialized model acting as an add-on to the pre-trained model that focuses on the TSC task. The pretext task chosen in this work is the following: given a set of M UTSCD, the task of the pre-trained model is to correctly predict from which dataset each sample belongs to. It is important to note that one could argue that a more intuitive approach is to combine all datasets and classes and predict a massive class distribution without the need of going through a pretext task. This last approach, however, would result in some issues when no correlation exists between classes of different datasets, so that the class distribution would not have a meaningful representation.

Once the pre-trained model is fully trained, the model is extended by a randomly initialized model. The new constructed model, made of a pre-trained

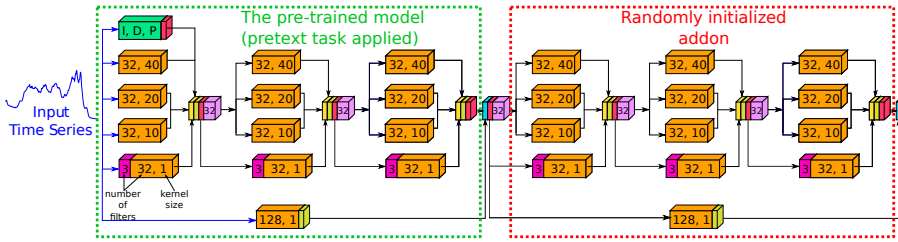

Fig. 2. The H-Inception architecture is divided into two sub-models: the pre-trained model, trained on the pretext task (dotted green rectangle), and the randomly initialized add-on model (dotted red rectangle). The H-Inception model is made of six Inception modules, each module containing three convolutional layers (in orange) and a MaxPooling layer (in magenta), followed by a concatenation (in yellow), a batch normalization layer (in oily), and an activation function (in red). Each Inception module, except the first one, is preceded by a bottleneck layer (in purple) to reduce the dimensionality and so the number of parameters. The first Inception module contains the hybrid addition, which is the hand-crafted convolution filter (in green). Residual connections do exist between the input and the third module, as well as between the third module and the output (in cyan).

and a randomly initialized sub-model, is then fine tuned on the TSC task for each dataset independently. In summary, the different steps of the whole training procedure are:

- Step 1: Given a set of M UTSCD datasets: $\{\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_{M-1}\}$, where $\mathcal{D}_i = \{(\mathbf{x}_j, \mathbf{y}_j)\}_{j=0}^{N_i-1}$, construct $\mathcal{D}_{PT} = \{(\mathbf{x}_n, \mathbf{yd}_n)\}_{i=0}^{N-1}$, where $N = \sum_{n=0}^{M-1} N_n$, is a dataset including all the time series from $\mathcal{D}_i$ with new labels $\mathbf{yd}$ that represent the dataset the input sample $\mathbf{x}$ belongs to;
- Step 2: Build a pre-trained model, $PT(.)$ with L_{PT} layers trained on $\mathcal{D}$ to correctly classify the dataset each sample belongs to;
- Step 3: Build, for each of the M datasets, a classifier $FT_i(.)$ for $i \in \{0, 1, \ldots, M-1\}$ with $L_{PT} + L_{FT}$ layers;
- Step 4: Fine tune a classifier $FT_i(.)$ for each dataset.

Backbone Model. In this work, we base our model on the-state-of-the-art deep learning model for TSC, the Hybrid Inception architecture (H-Inception) [6]. It is important to note that H-InceptionTime proposed in [6] is an ensemble of five H-Inception models trained with different intializations. For this reason, the backbone architecture in our approach is the H-Inception architecture, and we ensemble the trained models as well following the original work [6,12]. A summarized view of how the H-Inception backbone is decomposed into the pre-trained and fine tuning parts is presented in Figure 2. Given that the original H-Inception architecture is made of six Inception modules, the first three modules are set to be part of the pre-trained model and the last three are then added

to the fine tuning part. We refer to our approach using this specific H-Inception backbone as PHIT (pre-trained H-InceptionTime).

Batch Normalization Multiplexer (BNM). Most deep learning models for TSC [11] that achieve state-of-the-art performance on the UCR archive [2] are convolution-based architectures that also use the Batch Normalization layer with the goal of accelerating the training. In the H-Inception [6] backbone model that we chose, each convolution layer is followed by a Batch Normalization. The role of the Batch Normalization is to learn how to scale and shift the batch samples in order to get a zero mean and unit variance. However, this may be problematic when samples in a same batch are generated from different distributions, *i.e.*, from different datasets, such as in the case of our pre-trained model. For this reason, while training the pre-trained model on the pretext task, multiple Batch Normalization layers should be defined, one for each dataset, so as to replace the one usually used in modern CNN architectures for TSC. For this layer to work, we should then allow the model to connect each sample in the batch to the correct batch normalization layer. A visual representation of the proposed Batch Normalization Multiplexer (BNM) is presented in Figure 3. From the figure, it can be observed that the BNM takes as input the outcome of the previous layer, with the information of the dataset of the used series, being this information the same one the model is trying to predict. The dataset information goes through the control node of the BNM and chooses which Batch Normalization layer the output node should be connected to.

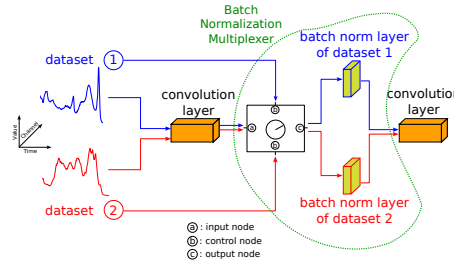

Fig. 3. The proposed BNM, is constituted of multiple batch normalization layers (in oily with blue and red contours) preceded by a multiplexer. This multiplexer has three nodes: (a) the input node, where the input time series goes through, (b) the control node, where the information about the dataset this input time series belong to goes through, and (c) the output node. The path selected for the output node is controlled by the node (b).

4 Results and Analysis

Datasets. To evaluate the performance of our proposed approach, we conducted a series of experiments on the UCR archive dataset [2], which comprises 128 datasets. However, due to redundancies in the archive, our study narrows it down to only 88 datasets. For instance, identical datasets appear multiple times but with varied train-test splits for distinct classification tasks. Such overlaps

could compromise the integrity of our model's training as it aims to predict the source dataset of a sample. Moreover, some datasets, while seemingly distinct, merely had varied class counts or were truncated versions of another. A detailed discussion of the reasons for excluding some datasets is reported in Table 1 of the supplementary materials. All datasets underwent a z-normalization prior to training to ensure a zero mean and unit variance. As samples from these datasets may differ in length, zero padding was applied within each batch (rather than before training) to align with the length of the longest series.

Division of the Datasets into Types. The purpose of using a pre-trained model is that of boosting the performance of the deep learning classifier on small datasets using knowledge learned on large ones. This is intuitively most applicable in the case where both the large and small datasets have at least basic information in common. For this reason, we do eight different pretext experiments following the number of dataset types that exist in the UCR archive. In particular, we used all of the datasets of the ECG type to train a pre-trained model, then fine tuned on each dataset independently. These eight types with the corresponding number of datasets are the following: Electrocardiogram (ECG) - 7 datasets, Sensors - 18 datasets, Devices - 9 datasets, Simulation - 8 datasets, Spectrogram - 8 datasets, Motion - 13 datasets, Traffic - 2 datasets, Images contour - 23 datasets.

Implementation Details. The proposed method is implemented in *Tensorflow python* and the code will be available upon acceptance. All of the parameters of the H-Inception model follow the same as in the original work [6]. Each experiment was performed with five different initializations, including the pre-trained and the fine tuned models. Results of multiple runs were assembled together and the model used for evaluation is the best model monitored during training following the training loss. We used a learning rate decay, ReduceLROnPlateau in *keras*, to reduce the learning rate during training by monitoring the train loss with a factor of half. All models were trained on a batch size of 64; the pre-trained model was trained for 750 epochs and the fine tuned model was trained for 750 epochs as well. This last condition ensured us to not train the model for more epochs than the baseline (*i.e.*, the baseline was trained for 1500 in [6]). All experiments were conducted on an Ubuntu 22.04 machine with an NVIDIA GeForece RTX 3090 graphic card with 24GB of memory.

4.1 Comparing Pre-Training with Baseline (Ensemble)

We present in this section a 1 vs. 1 comparison between our pre-training approach using the H-Inception architecture and the baseline. In what follows, we refer to our approach as Pre-Trained H-InceptionTime (PHIT).

Figure 4 represents this 1 vs. 1 comparison by a scatter plot between PHIT and H-InceptionTime. Each point represents a UCR dataset, where the x and y axis report the accuracy metric of H-InceptionTime and PHIT, respectively. The

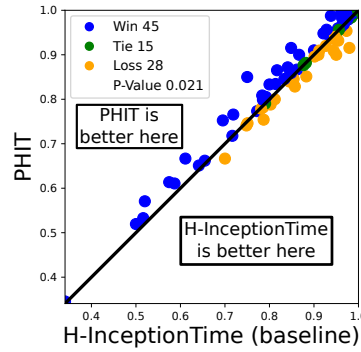

Fig. 4. A 1 vs. 1 scatter plot that compares H-InceptionTime (baseline) and PHIT using the accuracy metric. Each point represents a dataset, where the x and y axis represent the accuracy of H-InceptionTime and PHIT, respectively. A blue point represents a win for PHIT, an orange point a win for H-InceptionTime and a green point a tie.

accuracy is evaluated on the test set for each dataset using both methods. This 1 vs. 1 shown that over the 88 datasets, PHIT performs much better than the baseline. From the legend of Figure 4 it can be seen that PHIT wins 45 times over the baseline; the baseline wins only 28 times. To evaluate the statistical significance of this difference in performance, we presented as well a p-value produced using the Wilcoxon Signed-Rank Test. This p-value, represents the % of confidence of a difference in performance being statistically significant. If the p-value is less than 5% it means there is not enough datasets to conclude a statistical significance in the difference of performance. In this comparison, as seen in Figure 4, the p-value between PHIT and the baseline is almost 2.1%, which means PHIT significantly outperforms the baseline.

Table 1. The Win/Tie/Loss count between the proposed PHIT approach and the baseline (H-InceptionTime) per dataset domain. The first column presents the number of datasets included per domain followed by the number of Wins for PHIT, number of Ties, and number of Wins for the baseline. We include as well the percentage of number of losses and the average difference in accuracy (PHIT - baseline). A positive value in the last column indicates that on average of all datasets in a specific domain, PHIT performs better than the baseline on the accuracy metric (lowest value 0.0 and highest value 1.0).

Dataset Type	**Number of Datasets**	**Wins of PHIT**	**Ties of PHIT**	**Losses of PHIT**	Percentage of Losses	**Difference in Average Accuracy (PHIT - Baseline)**
Devices	9	4	0	**5**	**55.55 %**	+0.0046
ECG	7	**3**	2	2	**28.57 %**	+0.0012
Images	23	**14**	2	7	**30.43 %**	+0.0087
Motion	13	**11**	1	1	**07.69 %**	+0.0179
Sensors	18	**7**	5	6	**33.33 %**	+0.0002
Simulation	8	**3**	**3**	2	**25.00 %**	+0.0051
Spectro	8	**3**	2	**3**	**37.50 %**	+0.0115
Traffic	2	0	0	2	**100.0 %**	-0.0333

Analysing Performance per Domain. In Table 1, we present a detailed analysis on the performance of the proposed PHIT approach compared to the baseline per dataset domain. We present, for each domain used in the UCR archive, the total number of datasets and the Win/Tie/Loss count with the average difference in performance in the last column. A positive value in the last column confirms that on average PHIT outperforms the baseline on the average accuracy metric. We also present in the 5th column the percentage of number of losses of PHIT. From the table it can be seen that the percentage of losses never exceeds 50% more than twice, and that the average difference in performance is always positive except on one type (*Traffic*). These observations indicate that not only PHIT outperforms the baseline on a global scale of the UCR archive on the majority of domains. This comparison shows that fine tuning a pre-trained model on a generic task, which is in common between multiple datasets is significantly better than the traditional approach.

4.2 Visualizing the Filters

Since we base our work on CNNs, we can compare the space of the learned filters to see the effect of the pre-training approach. In order to visualize this space, we used the t-Distributed Stochastic Neighbor Embedding (t-SNE) visualization technique to reduce the dimensionality of the filters into a 2D plane [6]. By taking the filters of the first Inception module from the baseline, the pre-trained model and the fine tuned model, we can visualize the filters in Figure 5. In this figure, we consider the experiment over the ECG datasets, where we choose a couple: ECG200 and NonInvasiveFetalECGThorax1. We chose these two datasets given the difference in size of the training set. For instance, ECG200 has 100 training examples, whereas NonInvasiveFetalECGThorax1 has 1800.

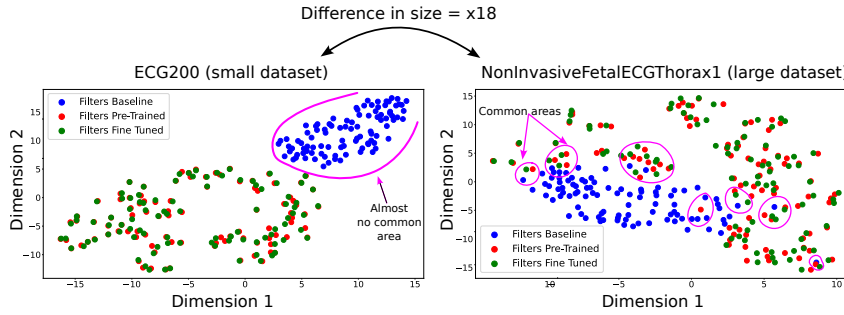

Fig. 5. A two dimensional representation of the filters coming from the first Inception module of the baseline (in blue), pre-trained(red) and fine tuned (green) models. The used datasets in this study are ECG200 (left) and NonInvasiveFetalECGThorax1 (right). The magenta areas represent the areas around the filters of the baseline model.

From Figure 5, the filters of the baseline, pre-trained and fine tuned models are presented for each dataset. The first noticeable aspect is that the blue points, representing the filters of the baseline, are quite different from the other red and green points. This ensures that by using the pre-trained model, then fine tuning it, the backpropagation algorithm learns different convolution filters than the traditional baseline approach. The second noticeable thing is that there exists a difference between both plots. On the one hand, in the case of ECG200 (left plot), almost no common areas exist between the filters of the three models. On the other hand, in the case of NonInvasiveFetalECGThorax1 (right plot) there exist many common areas between the filters of different colors. However, there exist some new areas for the pre-trained and fine tuned filters (green and red), which indicates that even though the dataset is large enough, the pre-trained model explored new filters given what it learned from other datasets.

4.3 Comparison with the State-of-the-Art

In what follows, we utilize a comparison technique proposed in [5] called the Multi-Comparison Matrix (MCM). This MCM presents a pairwise comparison between the classifiers as well as their ordering following the average performance. The MCM has shown to be stable to the addition and removal of classifiers, which gives it an advantage over other comparison approaches. The MCM presents as well the Win/Tie/Loss count and a $p-$value generated using the two tailed Wilcoxon Signed-Ranked Test to study the significance in the difference of performance. The MCM presents as well an ordering of performance of all classifiers following their average performance. In what follows, we present the MCM to compare PHIT to the state-of-the-art approaches including deep and non-deep learning approaches in Figure 6. It can be concluded that on the 88 datasets of the UCR archive, PHIT outperforms all of the deep learning approaches following the average performance metric. The MCM also shows that given the 88 datasets, no conclusion can be found on the statistical significance difference in performance between PHIT and the state-of-the-art MultiROCKET.

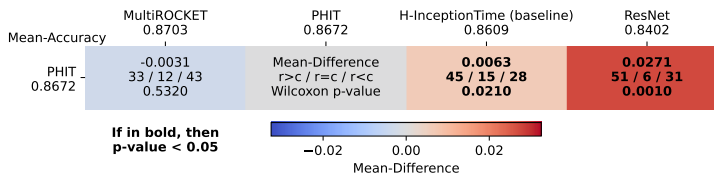

Fig. 6. A Multi-Comparison Matrix (MCM) representing the comparison between the proposed approach PHIT with the state-of-the-art approaches.

In order to also compare our approach with HIVE-COTE2.0 (HC2) [13] and Hydra+MultiROCKET (HydraMR) [3,14], we only used 86 datasets given that

for some datasets of the UCR archive the results are not provided on the original versions for these two models. The scatter plots showing the performance of PHIT compared to HC2 and HydraMR are presented in Figure 7. On the one hand, this figure shows that PHIT is still not as good as the HydraMR though the scatter plot shows that on 36 datasets, PHIT wins with a significant margin. On the other hand, no conclusion can be made on the statistical significance in the difference of performance between HC2 and PHIT. This concludes that the proposed approach is able to boost a lot the baseline deep learner to achieve HC2 state-of-the-art performance.

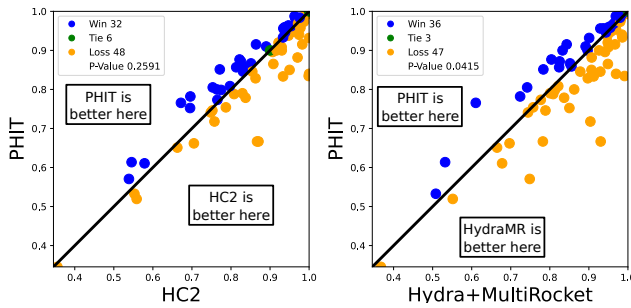

Fig. 7. Two 1 vs. 1 scatter plots representing the comparison between the proposed approach, PHIT, with two state-of-the-art models for TSC, HIVE-COTE2.0 (HC2) and HydraMultiROCKET (HydraMR).

5 Conclusion

In this work, we addressed the Time Series Classification problem by employing innovative pre-trained domain foundation models effectively mitigating overfitting issues in small datasets. Leveraging the UCR archive for evaluation, our methodology involved training models on multiple datasets to accurately classify each sample's original dataset. Subsequent fine-tuning of these models on individual datasets demonstrated superior performance over traditional methods, as evidenced by comprehensive experiments and analyses on the UCR datasets. Our contribution is the creation of domain-specific pre-trained foundation models for time series datasets in the UCR archive, offering a resource for researchers and paving the way for future extensions. This approach, with its inherent generic filters, holds promise for efficient adaptation to new datasets, potentially revolutionizing the training process in time series classification.

Acknowledgment

This work was supported by the ANR DELEGATION project (grant ANR-21-CE23-0014) of the French Agence Nationale de la Recherche. The authors would like to acknowledge the High Performance Computing Center of the University of

Strasbourg for supporting this work by providing scientific support and access to computing resources. Part of the computing resources were funded by the Equipex Equip@Meso project (Programme Investissements d'Avenir) and the CPER Alsacalcul/Big Data. The authors would also like to thank the creators and providers of the UCR Archive.

References

1. Ay, E., Devanne, M., Weber, J., Forestier, G.: A study of knowledge distillation in fully convolutional network for time series classification. In: Int. Joint Conf. on Neural Networks (IJCNN) (2022)
2. Dau, H.A., Bagnall, A., Kamgar, K., Yeh, C.C.M., Zhu, Y., Gharghabi, S., Ratanamahatana, C.A., Keogh, E.: The ucr time series archive. IEEE/CAA Journal of Automatica Sinica **6**(6), 1293–1305 (2019)
3. Dempster, A., Schmidt, D.F., Webb, G.I.: Hydra: Competing convolutional kernels for fast and accurate time series classification. Data Mining and Knowledge Discovery pp. 1–27 (2023)
4. Ismail-Fawaz, A., Devanne, M., Berretti, S., Weber, J., Forestier, G.: Lite: Light inception with boosting techniques for time series classification. In: Int. Conf. on Data Science and Advanced Analytics (DSAA) (2023)
5. Ismail-Fawaz, A., Dempster, A., Tan, C.W., Herrmann, M., Miller, L., Schmidt, D.F., Berretti, S., Weber, J., Devanne, M., Forestier, G., et al.: An approach to multiple comparison benchmark evaluations that is stable under manipulation of the comparate set. arXiv preprint arXiv:2305.11921 (2023)
6. Ismail-Fawaz, A., Devanne, M., Weber, J., Forestier, G.: Deep learning for time series classification using new hand-crafted convolution filters. In: IEEE Int. Conf. on Big Data (IEEE BigData). pp. 972–981 (2022)
7. Ismail-Fawaz, A., Devanne, M., Weber, J., Forestier, G.: Enhancing time series classification with self-supervised learning. In: Int. Conf. on Agents and Artificial Intelligence (ICAART) (2023)
8. Ismail-Fawaz, A., Ismail Fawaz, H., Petitjean, F., Devanne, M., Weber, J., Stefano, B., Webb, G., Forestier, G.: Shapedba: Generating effective time series prototypes using shapedtw barycenter averaging. In: ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data (2023)
9. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Data augmentation using synthetic data for time series classification with deep residual networks. In: ECML/PKDD Workshop on Advanced Analytics and Learning on Temporal Data (2018)
10. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Transfer learning for time series classification. In: IEEE Int. Conf. on Big Data (Big Data) (2018)
11. Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., Muller, P.A.: Deep learning for time series classification: a review. Data mining and knowledge discovery **33**(4), 917–963 (2019)
12. Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D.F., Weber, J., Webb, G.I., Idoumghar, L., Muller, P.A., Petitjean, F.: Inceptiontime: Finding Alexnet for time series classification. Data Mining and Knowledge Discovery **34**(6), 1936–1962 (2020)

13. Middlehurst, M., Large, J., Flynn, M., Lines, J., Bostrom, A., Bagnall, A.: Hive-cote 2.0: a new meta ensemble for time series classification. Machine Learning **110**(11-12), 3211–3243 (2021)
14. Middlehurst, M., Schäfer, P., Bagnall, A.: Bake off redux: a review and experimental evaluation of recent time series classification algorithms. arXiv preprint arXiv:2304.13029 (2023)
15. Wang, Z., Yan, W., Oates, T.: Time series classification from scratch with deep neural networks: A strong baseline. In: Int. Joint Conf. on Neural Networks (IJCNN) (2017)